h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbmn
FACHBEREICH MATHEMATIK
UND NATURWISSENSCHAFTEN

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

Hochschule Darmstadt

Fachbereiche Mathematik
und Naturwissenschaften
& Informatik

Datengetriebener Schema-Evolution-Prozess zur Erkennung von Multitype-Schema-Operationen in NoSQL-Datenbanken

Abschlussarbeit zur Erlangung des akademischen Grades
Master of Science (M. Sc.)
im Studiengang Data Science

vorgelegt von

Dominik Ludwig

Matrikelnummer: 739538

Referentin	:	Prof. Dr. Uta Störl
Korreferent	:	Prof. Dr. Peter Muth

Ausgabedatum	:	13.07.2020
Abgabedatum	:	28.12.2020

ERKLÄRUNG

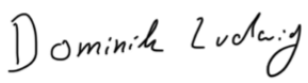
Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Frankfurt am Main, 24. Dezember 2020



Dominik Ludwig

GEHEIMHALTUNGSERKLÄRUNG

Diese Masterarbeit darf weder vollständig noch auszugsweise ohne schriftliche Zustimmung des Autors vervielfältigt, veröffentlicht oder Dritten zugänglich gemacht werden. Die abgegebenen Masterarbeiten werden von dem Hauptreferenten unter Verschluss gehalten.

Mir ist bekannt, dass die Geheimhaltung nicht die Erstellung eines Masterposters sowie die Durchführung des Kolloquiums berührt. Die Geheimhaltungsverpflichtung erlischt automatisch nach fünf Jahren.

Frankfurt am Main, 24. Dezember 2020

Dominik Ludwig

ABSTRACT

The principles of agile working methods, short development cycles, customer proximity and customer understanding have arrived in data modeling. Changes to the data model should be possible at any point in the development cycle. A change in a data model directly affects data consumers. The development of a data model takes place independently of the application development. This is taken to extremes by a data lake architecture. The database developer on the one hand works with many changes to the data model. On the other hand, the application developer expects a stable interface to the same data. This creates tension because changes to the data model, especially at high frequency, cannot lead to garantee an stable interface. In order to bridge this tension, a schema evolution process is introduced.

Evolution is an continuous improvement process and passing advantageous properties to a child generation of a living being. Analogous to this, advantageous properties are passed to later versions of the same data model in the schema evolution process. Schema evolution processes accompany the schema evolution in such a way that schema changes are made orderly manner and known to the data consumers. Multitype schema operations are cross-entity schema operations. These are caused by copying or moving attributes. Data consumers are interested in multitype schema operations, since such schema changes represent a semantic change of an existing interface.

In this thesis a data-driven schema evolution process for a data-lake architecture using the NoSQL database Apache Hive has been prototypically developed in cooperation with Commerzbank AG. By continuous data analysis, schema changes from data modeling can be detected. Schema evolution is not a theoretical construct. Within one month up to 2.500 schema changes registered at Commerzbank. This high number motivates the need of an automated schema evolution process. Data analysis includes the steps of similarity calculation, compound calculation and schema operation extraction. Similarities are used to identify attribute pairs that are affected by an evolution process. A composite calculation establishes a row relationship between those attributes. Schema operations can then be extracted by analysing the schema history. Finally, multitype schema operations can be obtained from the schema history.

Three data models are used to evaluate correctness, completeness and performance. The TPC-H data model describes a generic data warehouse scenario. In addition, the data models *CORE* and *KK-Buchung* from the Commerzbank are considered. To evaluate the correctness of the procedures, the generated result is compared with the expected result according to the data model specification. Simulated multitype schema operations in the data models can be used to confirm the correctness. By a detailed status logging the complete processing can be ensured. A variation of the number of entities leads to a linear increase in runtime. Thus, the presented methods are suitable for a practical usage.

Keywords schema evolution, similarity measures, NoSQL databases

ZUSAMMENFASSUNG

Die Prinzipien der agilen Arbeitsweise, kurze Entwicklungszyklen, Kundennähe und Kundenverständnis sind in der Datenmodellierung angekommen. Änderungen am Datenmodell sollen zu jedem Zeitpunkt im Entwicklungszyklus möglich sein. Haupttreiber solcher Änderungen sind Anwendungen, die Daten konsumieren. Eine Veränderung eines Datenmodells beeinflusst Datenkonsumenten. Die Entwicklung eines Datenmodells findet unabhängig von der Anwendungsentwicklung statt. Dies wird durch eine Data-Lake-Architektur auf die Spitze getrieben. Der Datenbankentwickler auf der einen Seite arbeitet mit vielen Änderungen am Datenmodell. Auf der anderen Seite erwartet der Anwendungsentwickler eine stabile Schnittstelle zu denselben Daten. Das führt zu einem Spannungsverhältnis, weil Änderungen des Datenmodells, zumal in hoher Frequenz, nicht dazu führen können, dass die Schnittstelle ständig unerwartete Ergebnisse liefert. Um dieses Spannungsverhältnis zu überbrücken, wird ein Schema-Evolutions-Prozess eingeführt.

Evolution ist ein kontinuierlicher Verbesserungsprozess sowie die Weitergabe vorteilhafter Eigenschaften an eine Kindgeneration eines Lebewesens. Analog dazu werden in der Schema-Evolution vorteilhafte Eigenschaften an spätere Versionen desselben Datenmodells weitervererbt. Schema-Evolutions-Prozesse begleiten die Schema-Evolution dergestalt, dass Änderungen am Schema den Konsumenten der Daten in einem geordneten Verfahren bekannt gemacht werden.

Multitype-Schema-Operationen sind entitätsübergreifende Schema-Operationen. Solche werden durch ein Kopieren oder Verschieben von Attributen verursacht. Datenkonsumenten sind an Multitype-Schema-Operationen interessiert, da solche Schema-Veränderungen eine semantische Veränderung einer bestehenden Schnittstelle darstellen.

In Zusammenarbeit mit der Commerzbank AG ist ein datengetriebener Schema-Evolutions-Prozess für eine Data-Lake-Architektur unter der Verwendung der NoSQL-Datenbank Apache Hive prototypisch entwickelt worden. Durch eine kontinuierliche Datenanalyse können Schema-Veränderungen erkannt werden. Das Thema Schema-Evolution ist kein theoretisches Konstrukt. Innerhalb eines Monats werden bei der Commerzbank in etwa 2.500 Schema-Veränderungen registriert. Diese hohe Anzahl motiviert die Notwendigkeit nach einem automatisierten Schema-Evolutions-Prozess. Die Datenanalyse umfasst die folgenden Schritte: Ähnlichkeitsberechnung, Verbundberechnung und Schema-Operations-Extraktion. Mithilfe von Ähnlichkeiten werden Attribute identifiziert, die durch einen Evolutionsprozess betroffen sind. Durch eine Verbundberechnung wird zwischen diesen Attributen eine Zeilenbeziehung hergestellt. Abschließend können Multitype-Schema-Operationen aus der Schema-Historie gewonnen werden.

Anhand von drei Datenmodellen werden die Korrektheit, Vollständigkeit und die Performance des Schema-Evolutions-Prozesses evaluiert. Das TPC-H-Datenmodell beschreibt ein generisches Datawarehouse-Szenario. Zusätzlich werden die Datenmodelle *CORE* und *KK-Buchung* der Commerzbank betrachtet. Für die Evaluation der Korrektheit der Verfahren werden die Ergebnisse mit den Datenmodellspezifikationen verglichen. Anhand von simulierten Multitype-Schema-Operationen kann die Korrektheit der Verfahren bestätigt werden. Mit einer ausführlichen Statusprotokollierung kann eine vollständige Datenverarbeitung sichergestellt werden. Eine lineare Steigerung der Anzahl der Entitäten führt zu einer linearen Laufzeitsteigerung. Somit eignen sich die vorgestellten Verfahren für einen produktiven Einsatz.

Keywords Schema-Evolution, Ähnlichkeitsmaße, NoSQL-Datenbanken

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

ABKÜRZUNGSVERZEICHNIS

API	Application-Programming-Interface
DBMS	Datenbankmanagementsystem
JSON	JavaScript-Object-Notation
HDFS	Hadoop-Distributed-File-System
SQL	Structured-Query-Language
JDBC	Java-Database-Connectivity
DSL	Domain-Specific-Language
CDF	Cumulative-Distribution-Function
OLTP	Online-Transaction-Processing
TPC	Transaction-Processing-Performance-Council
DSGVO	Datenschutz-Grundverordnung
CPU	Central-Processing-Unit
QoS	Quality-of-Service
MQTT	Message-Queuing-Telemetry-Transport
KS	Kolmogorow-Smirnow
KL	Kullback-Leibler-Divergenz
JIA	Jaccard-Index-Alphabetisch
JIN	Jaccard-Index-Numerisch
TP	True-Positive
FP	False-Positive
FN	False-Negative
TN	True-Negative
RHS	Right-Hand-Side
LHS	Left-Hand-Side
FD	Funktionale Abhängigkeit
IND	Inklusionsabhängigkeiten
UCC	Unique-Column-Combination
NoSQL	Not-only-SQL
UDF	User-Defined-Function

Teil I

THESIS

1 EINFÜHRUNG

1.1 MOTIVATION UND ZIELE

Die Modellierung von Daten ist keineswegs statisch. Veränderte fachliche Anforderungen an ein Datenmodell, aber auch das Korrigieren fehlerhafter Modelle führen dazu, dass die Datenmodelle Gegenstand kontinuierlicher Veränderung sind. Jede Anpassung kann eine Kaskade von weiteren Anpassungen in anderen Datenmodellen verursachen. Im Kontext einer Data-Lake-Architektur[1] ist dies durch die potenziell hohe Anzahl an Datenkonsumenten eine Herausforderung.

Zwischen Datenproduzenten und Datenkonsumenten wird eine Schnittstelle in Form eines Datenmodells aufgebaut. Jede Veränderung eines Datenmodells birgt die Gefahr einer Verletzung dieser Schnittstelle. Anwendungen, die als Datenkonsumenten agieren, sind durch diese Verletzungen betroffen und müssen jene Schnittstellenveränderungen berücksichtigen. Aus diesem Grund wird ein autonom agierender Prozess benötigt, der Veränderungen an Datenmodellen erkennt und in dessen Rahmen Schema-Operationen extrahiert werden können. Alle Datenkonsumenten sind an diesen Schema-Operationen interessiert und können ihre Datenmodelle entsprechend anpassen. Im Weiteren Verlauf wird dieser Prozess als Schema-Evolutions-Prozess bezeichnet, da eine kontinuierliche Schema-Veränderung einem natürlichen Evolutionsprozess ähnelt.

Die Hauptbetrachtung im Schema-Evolutions-Prozess gilt dem Schema. Im Kontext von NoSQL-Datenbanken wird ein Schema durch eine Menge an Entitäts-Typen definiert. Ein Entitäts-Typ ist die Beschreibung einer Entität und umfasst dessen Struktur. Eine Entität ist eine Realisierung eines Entitäts-Typen und beinhaltet Daten [37]. Die Möglichkeit, ein Schema zu definieren, ist keine Grundvoraussetzung für eine NoSQL-Datenbank [34][vgl. 2]. Mithilfe eines datengetriebenen Ansatzes können Schemas nachträglich erzeugt werden. Dieser Schritt ermöglicht es, unabhängig von einer Schema-Definition durch einen Datenbankhersteller einen Schema-Evolutions-Prozess zu etablieren.

In einem Schema-Evolutions-Prozess können Singletype- und Multitype-Schema-Operationen unterschieden werden. Eine Singletype-Schema-Operation führt zu einer Schema-Veränderung einer einzelnen Entität. Wenn ein Attribut im Zuge einer Schema-Veränderung von einer Entität in eine andere verschoben oder kopiert wird, wurde eine Multitype-Schema-Operation durchgeführt. Für die Erkennung von Multitype-Schema-Operationen im Schema-Evolutions-Prozess ist eine entitätsübergreifende Datenanalyse notwendig, da nicht nur Attribute, sondern auch Entitätsbeziehungen identifiziert werden müssen. Datenkonsumenten sind an Multitype-Schema-Operationen besonders interessiert, da solche Schema-Veränderungen semantische Veränderungen mehrerer Entitäten darstellen.

Es wird zwischen der Data-Discovery und der Data-Migration im Schema-Evolutions-Prozess unterschieden. Mithilfe der Data-Discovery werden Schema-Operationen in Datenquellen aufgedeckt [18][vgl. 2380]. Im Anschluss kann durch die Data-Migration eine konsolidierte Datensicht durch das Anwenden der Schema-Operationen erreicht werden [19][vgl. 2764]. In dieser Arbeit liegt der Fokus in der Data-Discovery von Schema-Operationen.

1 Ein Data-Lake umfasst konsolidierte Datenbestände unterschiedlicher Unternehmensbereiche und kann als eine zentrale Datenquelle eines Unternehmens angesehen werden. Potenzielle Datenkonsumenten können über ausgewiesene Schnittstellen Unternehmensdaten beziehen [13][vgl. 13].

Mengenwertige Attribute können in der Datenmodellierung definiert werden, zum Beispiel vom Typ Liste oder Map [23][vgl. 10]. Diese Datentypen werden nicht diskutiert, da der Fokus auf dem Schema-Evolutions-Prozess liegt.

Ziel dieser Arbeit ist es, durch die Etablierung eines Schema-Evolutions-Prozesses die Auswirkungen von Schnittstellenveränderungen für Datenkonsumenten zu erkennen. Im Fokus stehen die Multitype-Schema-Operationen. Die Multitype-Schema-Operationen müssen aus den Daten extrahiert werden können. Eine alleinige Betrachtung von Schema-Informationen ist nicht ausreichend. Neben dem Extrahieren der Schema-Operation ist die Vollständigkeit ein wichtiges Merkmal für das Nachvollziehen der Schema-Historie. Die Verfahren sind allgemeingültig zu halten. Spezifische Implementierungen eines Datenbankherstellers sind nicht zu verwenden.

1.2 STRUKTUR DER ARBEIT

In Kapitel 2 werden Verfahren und Begrifflichkeiten eingeführt. Die Unterscheidung zwischen einer relationalen und einer NoSQL-Datenbank wird beschrieben. Diese Unterscheidung motiviert den Einsatz von Domänenklassen. Mit einer Domänenklassifikation können Schemas datengetrieben erzeugt werden. Diese Schemas werden im Schema-Evolutions-Prozess betrachtet. Zu den Kernkomponenten des Schema-Evolutions-Prozesses gehört die Ähnlichkeits- und Verbundberechnung. Für diese Berechnungen werden Funktionseigenschaften definiert, die durch ausgewählte Funktionen erfüllt werden können. Im Kapitel 3 wird das Zusammenspiel aus Ähnlichkeits- und Verbundberechnungen für einen Schema-Evolutions-Prozess beschrieben. Durch eine Analyse der Schema-Historie können Multitype-Schema-Operationen extrahiert werden. Die Vorgehensweise wird an einem Beispiel demonstriert. Abschließend wird geklärt, ob ein Einsatz rechtlich möglich ist. Eine prototypische Implementierung wird in Kapitel 4 vorgestellt und in Kapitel 5 abschließend evaluiert.

1.3 ABGRENZUNG

Die Notwendigkeit eines Schema-Evolutions-Prozesses ist nicht erst seit dem Aufkommen von NoSQL-Datenbanken gegeben. Das Sicherstellen einer stabilen Datenschnittstelle zwischen Datenbankentwickler und Datenkonsumenten besteht auch für relationale Datenbanken. In [39] wird ein Schema-Evolutions-Prozess für eine relationale Datenbank beschrieben. Eine Datensicht kapselt Schema-Veränderungen vor einem Datenkonsumenten. Schema-Veränderungen können durch einen Datenbankentwickler ohne eine Beeinflussung der Datenkonsumenten durchgeführt werden. Schema-Veränderungen können nur erkannt werden, wenn ein Schema definiert ist. Dies ist durch die Eigenschaft der Schemaflexibilität von NoSQL-Datenbanken eine Herausforderung.

Eine Kernkomponente des Schema-Evolutions-Prozesses ist das Schema-Matching. Mit diesem können Beziehungen zwischen Schemas gefunden werden. Nach [28] werden folgende Schema-Matching-Verfahren unterschieden:

- Der **Schema-Only-Based-Matcher** berücksichtigt für das Verbinden von Schemas alleinig die Schema-Informationen, zum Beispiel den Namen oder den Datentyp eines Attributs. Ein Datenzugriff erfolgt nicht [28][vgl. 5].
- Der **Instance/Content-Based-Matcher** ist ein datengetriebener Ansatz für das Verbinden von Schemas. Dieser findet Anwendung, wenn keine oder unzureichende Schema-Informationen vorhanden sind [28][vgl. 9].

- Der **Hybrid-Matcher** berücksichtigt sowohl Schema-Informationen als auch die Daten für das Verbinden von Schemas. Über eine Schema-Analyse können Kandidaten für einen Datenzugriff gefunden werden. Die Qualität des Schema-Matchings kann so gesteigert werden [28][vgl. 10].

Für eine breite Abdeckung von NoSQL-Datenbanken werden in dieser Arbeit keine Schema-Informationen verwendet, die durch eine NoSQL-Datenbank bereitgestellt werden. Stattdessen werden Schemas datengetrieben erhoben. Es können verschiedene Verfahren zur Erzeugung von Schemas angewendet werden. In [3] wird ein Schema durch eine Analyse von JSON-Dateien erzeugt. Die Fokussierung auf ein spezielles Dateiformat wird durch die Heterogenität der Dateiformate in NoSQL-Datenbanken abgelehnt [23][vgl. 208]. Es werden datengetriebene Verfahren für ein nachträgliches Erzeugen von Schemas benötigt, die unabhängig von der Datendarstellung sind. In [18] wird ein Schema durch eine Strukturextraktion erzeugt. Grundlage dieser Extraktion sind die Metadaten, die durch eine NoSQL-Datenbank bereitgestellt werden. NoSQL-Datenbanken unterscheiden sich unter anderem durch den Umfang dieser bereitgestellten Metadaten. In dieser Arbeit werden für die Erzeugung eines Schemas die Entitätsnamen und die Attributsnamen jener Entitäten von einer NoSQL-Datenbank vorausgesetzt. Zusätzliche Metadaten müssen datengetrieben erhoben werden. In [8] wird dies durch das Einführen von Domänenklassen erreicht. Eine Domänenklasse ist eine mathematische Beschreibung von Daten. Für das Schema-Matching wird in dieser Arbeit ein Schema datengetrieben erzeugt. Dies ermöglicht einerseits eine hohe Abdeckung verschiedener NoSQL-Datenbanken, andererseits kann der Suchraum für das Verbinden von Schemas reduziert werden. Der Suchraum definiert die Anzahl der zu überprüfenden Mengen für das Schema-Matching. Die verbleibenden Mengen sind durch einen Instance-Based-Matcher auf eine Beziehung zu untersuchen. Durch die Berücksichtigung des eigens erzeugten Schemas und einer Datenanalyse wird ein Hybrid-Matcher in dieser Arbeit vorgestellt.

Der Fokus dieser Arbeit liegt in der Erkennung von Multitype-Schema-Operationen im Schema-Evolutions-Prozess. Ein solcher Prozess wird auch von [18] beschrieben. Nachdem ein Schema aus den Metadaten einer NoSQL-Datenbank generiert wurde, werden Schema-Veränderungen erkannt und mit vorherigen Schema-Versionen verglichen. Diese Differenzmenge zwischen der aktuellen und der vorherigen Schema-Version wird für das Ableiten von Singletype- und Multitype-Schema-Operationen verwendet. Für Multitype-Schema-Operationen wird ein Verbund zwischen zwei Entitäten benötigt. Ein Verbund beschreibt, wie Entitäten in eine semantische Beziehung gebracht werden können. Dieser Verbund wird durch die Berechnung von Inklusionsabhängigkeiten (IND) zwischen zwei Entitäten erreicht. Der Suchraum kann durch Optimierungsstrategien reduziert werden. Zu diesen zählen die Berücksichtigung des Datentyps und der Statistiken eines Datensatzes [18][vgl. 2467]. In dieser Arbeit wird ein Schema aus den Daten erzeugt. Beziehungen in den Daten werden durch Ähnlichkeitsmaße gefunden. Dieses Vorgehen unterscheidet sich wesentlich von [18][vgl. 2464], da der Fokus auf den Daten und nicht auf den bereitgestellten Schemas einer NoSQL-Datenbank liegt. Für das Verbinden von Schemas werden INDs betrachtet.

Abseits des Schema-Evolutions-Prozesses beschreiben [25] und [7], wie zwischen Datensätzen eine Ähnlichkeitsbeziehung gefunden werden kann. Diese Verfahren werden in dieser Arbeit im Schema-Evolutions-Prozess für das Erkennen von Ähnlichkeitsbeziehungen nach einer Schema-Veränderung verwendet.

Eine Softwarelösung für die Etablierung eines Schema-Evolutions-Prozesses ist *Hackolade* [14]. *Hackolade* ist eine Reverse- und Forward-Engineering Anwendung für NoSQL-Datenbanken. Nach [14] kann die Notwendigkeit von *Hackolade* folgendermaßen beschrieben werden:

> „Hackolade combines the graphical representation of collections in an Entity Relationship Diagram, with the graphical representation of the JSON Schema definition of each collection in a Tree View. Together, these graphical representations provide the schema model of a NoSQL application, and the documentation of that model. [..] The software facilitates the work of, and the dialog between analysts, architects, designers, developers, testers, DBAs, and operators of systems that are based on such technologies.“ [14]

Hackolade dokumentiert Veränderungen der Schnittstelle, die zwischen Datenbankentwickler und Datenkonsumenten vereinbart wurde. Im Schema-Evolutions-Prozess hingegen agieren Datenbankentwickler und Datenkonsument unabhängig voneinander. Es wird davon ausgegangen, dass eine Schema-Veränderung nicht durch eine externe Anwendung modelliert und anschließend verfügbar gemacht wird. Schema-Veränderungen müssen aus den Daten gewonnen werden können. Dieser Prozess muss automatisiert erfolgen, da mit vielen Schema-Veränderungen gerechnet werden muss [39][vgl. 6]. Das Erkennen von Schema-Operationen steht nicht im Fokus von *Hackolade*, da diese eine Datenanalyse voraussetzen [14]. In dieser Arbeit wird ein autonom agierender Schema-Evolutions-Prozess vorgestellt, der unabhängig von der Datenmodellierung Schema-Operationen durch eine Datenanalyse erkennen kann.

2 GRUNDLAGEN

In diesem Kapitel werden die Grundlagen für das Verständnis dieser Arbeit gelegt. Dazu gehört die Unterscheidung zwischen einer relationalen Datenbank zu einer Not-only-SQL (NoSQL)-Datenbank. Dies ist notwendig, da die Eigenschaften von NoSQL-Datenbanken den Schema-Evolutions-Prozess maßgeblich prägen. Das Konzept der Schema-Evolution wird anhand eines übersichtlichen Datenmodells erläutert. Wesentliche Teile des Schema-Evolutions-Prozesses sind: Domänenklassifikation sowie Ähnlichkeits- und Verbundberechnung. Diese Verfahren werden im Folgenden eingeführt.

2.1 DEFINITIONEN

2.1.1 *Relationale Datenbanken*

Eine relationale Datenbank folgt den Prinzipien der relationalen Algebra. Ein Attribut A_i ist die kleinste zu definierende Einheit einer Datenbank und entstammt dem Universum U. Der Wertebereich D_i eines Attributs wird durch die Definition einer Domäne $dom(A_i)$ eingegrenzt. Alle Attributswerte A_i müssen in $dom(A_i)$ enthalten sein. Ein Relationenschema $R \subseteq U$ ist eine Bündelung von Relationen. Eine Relation r ist durch das Relationenschema $r(R)$ bestimmt. Eine Relation kann $A_1 \dots A_n, n \in \mathbb{N}$ Attribute beinhalten. Ein Tupel t einer Relation ist die Vereinigung der Attribute $A \in R$. Ein Datenbankschema ist eine Menge von Relationenschemas $S = \{R_i \dots R_m\}, m \in \mathbb{N}$. Eine Datenbank ist eine Menge von Relationen, die durch ein Datenbankschema eingegrenzt wird [30][vgl. 8]. Ein wichtiges Konstrukt der relationalen Algebra ist das Verbinden von Relationen. So entsteht ein Netz von Relationen die durch ausgezeichnete Attribute zusammengeführt werden können. Beziehungen zwischen Relationen werden durch Primär- und Fremdschlüssel abgebildet. Ein Fremdschlüssel ist einem Primärschlüssel einer anderen Relation zugeordnet. Ein Fremdschlüssel X der Relation r_1 ist eine echte Obermenge zu einem Primärschlüssel Y der Relation r_2: $\{t(X)|t \in r_1\} \subseteq \{t(Y)|t \in r_2\}$. Für das Sicherstellen der Datenintegrität können Integritätsbedingungen definiert werden [30][vgl. 8].

2.1.2 *NoSQL-Datenbanken*

Die Begriffe der relationalen Algebra müssen für den Einsatz von NoSQL-Datenbanken angepasst werden. Durch das Konzept der Schemaflexibilität in NoSQL-Datenbanken kann kein Relationenschema definiert und somit kann eine Datenbank im relationalen Sinne nicht beschrieben werden [23][vgl. 202]. Stattdessen wird der Begriff der Entität eingeführt. Eine Entität e_i ist eine Zusammenfassung von Attributen A_i. Ein Attribut wird durch keinen datenbankseitig definierten Wertebereich einer Domäne eingeschränkt. Die Schemaflexibilität ermöglicht einerseits, dem starren Konzept der relationalen Struktur zu entkommen, andererseits müssen Integritätsbedingungen anderweitig, zum Beispiel durch eine Validierung innerhalb einer Anwendung, sichergestellt werden [23][vgl. 113]. Zusätzlich ist das Konzept des Schema-On-Read in NoSQL-Datenbanken verbreitet [34][vgl. 2]. Wenn Datensätze einer Entität hinzugefügt werden, wird nicht auf die Konformität eines Schemas geachtet. Die Daten werden in einem Datenspeicher abgelegt. Anschließend können diese durch die De-

finition eines Schemas gelesen werden. Diese Eigenschaft steht diametral zu dem Konzept Schema-On-Write, bekannt aus relationalen Datenbanken, da beim Hinzufügen von Daten auf die Einhaltung einer Schema-Definition geachtet wird [34][vgl. 2]. Weitere Charakteristika von NoSQL-Datenbanken sind unter anderem die Skalierung und eine differenzierte Sicht auf die transaktionalen Eigenschaften[1] [34][vgl. 139]. In dieser Arbeit liegen die Daten und deren Beziehungen im Vordergrund. Daher werden diese Charakteristika nicht weiter betrachtet.

2.2 DATENGRUNDLAGE

Für das Veranschaulichen der vorzustellenden Verfahren wird ein übersichtliches und vereinfachtes Datenmodell eingeführt. Das Datenmodell besteht aus den Entitäten *Player* und *Mission* [19][vgl. 2765]. Ein *Player* ist ein Interessent an einem Spiel und kann Missionen bewältigen. Es besteht eine 1:n Beziehung zwischen *Player* und *Mission*. Der Verbund wird über einen Fremdschlüssel in der *Player*-Entität (pid) mit einem Primärschlüssel der *Mission*-Entität (id) definiert. Einem Datenbankmanagementsystem (DBMS) stehen diese Informationen nicht zur Verfügung und müssen aus den Daten ermittelt werden. Es sind unterschiedliche Datentypen, wie zum Beispiel Zeichenketten, Zahlen und Daten in den Entitäten vertreten. In Abbildung 2.1 sind die genannten Entitäten zuzüglich Attribute und Datentypen gezeigt.

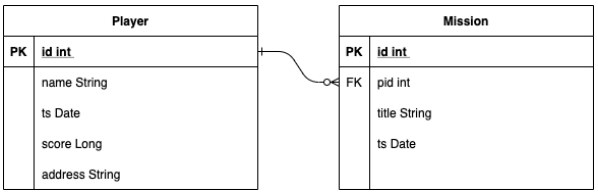


Abbildung 2.1: Vereinfachtes Datenmodell für den Schema-Verbund nach [19][vgl. 2765]

An einen Testdatensatz werden zwei Anforderungen gestellt:

Erste Anforderung: Es werden unterschiedliche Ansätze für die Berechnung von Ähnlichkeiten in den Daten vorgestellt. Somit werden Datensätze benötigt, deren Verfremdungsgrad gesteuert werden können. Im Schema-Evolutions-Prozess ist dies notwendig, da durch eine Schema-Operation ein Attribut in eine andere Entität verschoben werden kann. Nun existieren zwei Entitäten, die je einen Teil des Attributs abdecken. Ein Ähnlichkeitsmaß wird benötigt, da die Datenbasis beider Entitäten unterschiedlich sein können. So kann die ursprüngliche Entität die gesamte Historie eines Datensatzes beinhalten. Durch die Schema-Operation erhält die zweite Entität nur die neu hinzugekommenen Datensätze. Es wird nicht angenommen, dass jene Entitäten nach vollzogener Schema-Operation identisch sind. Daher muss dies mithilfe eines Ähnlichkeitsmaßes ausgeglichen werden.

In [20][vgl. 8] werden unterschiedliche Verfahren für ein künstliches Verfremden von Daten beschrieben. Für jedes alphabetische Attribut wird eine zufällige Kaskade von Veränderungsoperationen durchgeführt. Es lässt sich durch eine Variation der Kaskade der Verfremdungsgrad variieren[2].

1 Für NoSQL-Datenbanken wird eine flexiblere Sicht auf Transaktionen benötigt. BASE (**B**asically-**A**vailable, **S**oft-state und **E**ventual-Consistency) anstelle von ACID (**A**tomicity, **C**onsistency, **I**solation und **D**urability). Durch die BASE-Eigenschaften wird beschrieben, dass der Fokus nicht auf sofortiger Datenkonsistenz liegt, sondern auf der Verfügbarkeit und Skalierung der Daten[30][vgl. 639].

2 Die Ausführung ist dann deterministisch, wenn ein *seed*-Wert für die Initialisierung des Zufallszahlengenerators gesetzt wird.

- **add**: Innerhalb einer Zeichenkette wird an einer zufällig zu bestimmenden Position ein zusätzliches Zeichen hinzugefügt. Das Zeichen ist aus einer vordefinierten Menge, zum Beispiel $[A-Za-z]$[3], zu bestimmen.
- **remove**: Analog zum Hinzufügen eines Zeichens wird in diesem Verfahren ein Zeichen entfernt.
- **replace**: An einer zufällig zu bestimmenden Position wird ein Zeichen ersetzt.
- **swap**: Es werden zwei zufällig zu bestimmenden Positionen innerhalb einer Zeichenkette ermittelt. Die Werte an diesen Positionen werden vertauscht.
- **permut**: Innerhalb einer Zeichenkette werden mehrere Positionen bestimmt. Die betroffenen Zeichen werden zufällig vermischt und in den jeweiligen Zeichenpositionen wieder eingefügt.

Für numerische Attribute können die vorgestellten Verfahren nicht angewendet werden, da eine Modifikation eines Wertes an einer beliebigen Position die natürliche Ordnung der Attributswerte zerstört [20][vgl. 8].

- Für diskrete und stetige Werte, ohne eine natürliche Ordnung, lässt sich eine Verfremdung durch das Hinzufügen eines zufälligen Wertes aus einer vordefinierten Verteilung erreichen. Es ist zu überprüfen, ob der so entstandene Wert den möglichen Wertebereich über/unterschreitet.
- Für ordinal und metrisch skalierte Werte werden Stichproben innerhalb des Wertebereichs erzeugt. Eine Veränderung der Datenverteilung ist nicht vorgesehen.
- Primär- und Fremdschlüssel sind nicht zu verfremden. Für ein Schlüsselattribut ist die Identitätsfunktion zu verwendet.

Das Attribut *score* der *Player*-Entität gehört zu den ordinal[4] skalierten Werten. Innerhalb des Wertebereichs wird ein neuer Wert zufällig gezogen und dem Attribut zugewiesen. Für sonstige numerische Attribute wird eine zufällige Varianz aus einer vordefinierten Verteilung addiert. Es wird eine Normalverteilung mit den Parametern $N(\mu = 0, \sigma = 0.5)$ gewählt[5]. Durch das Anwenden der Veränderungsoperationen kann für die *Player*-Entität folgende Attributswerte ermittelt werden:

Verfahren	**ts**	**name**	**score**	**address**
Ursprung	1588632462	Quinn Graves	22	Postfach 127, Suspendisse Ave
Alphabetische Verfremdung	1588632462	**QuirG nnaves**	22	**Postfach 127, Susnipsesde Ave**
Numerische Verfremdung	**1588632473**	QuirG nnaves	**55**	Postfach 127, Susnipsesde Ave

Tabelle 2.1: Verfremdung eines Player-Tupels

Zweite Anforderung: Eine Entitätsbeziehung muss aus den Daten extrahiert werden können. Primär- und Fremdschlüssel müssen vor der Ausführung der Verfahren bekannt sein. Für eine aussagekräftige Validierung müssen mehrere Entitäten in dem Datensatz vorhanden sein. In dem Testdatensatz sind die Entitäten *Player* und *Mission* miteinander verbunden. Attributswerte können durch entsprechende Testdatengeneratoren[6] bezogen werden. Die Semantik und die Domäne eines Attributs müssen diesen Generatoren mitgeteilt werden. Für das *Mission-Player*-Datenmodell werden jeweils zwei Entitäten erzeugt. Die originalen Entitäten sind in den Tabellen 2.2 und 2.4 gezeigt. Durch das Anwenden der Verfremdungsverfahren aus [20]

3 Für die Definition der erlaubten Zeichenmenge wird ein regulärer Ausdruck verwendet. In diesem Fall sind alle Zeichen von *a* bis *Z* erlaubt.

4 Eine ordinal skaliertes Attribut besitzt eine endliche Anzahl an Ausprägungen. Eine Interpretation der Abstände zwischen den Ausprägungen ist nicht möglich/sinnvoll.

5 Die Wahl der Parameter für μ und σ sind frei wählbar, da die Verteilung für ein zufälliges Rauschen in den Daten verwendet wird, sollten die Werte um $\mu = 0$ mit einer Streuung von $\sigma = 0.5$ liegen, da ansonsten die Variationen in dem vorliegenden Datensatz zu hoch wäre.

6 Es wurde der Testdatengenerator der Webseite: `http://www.generatedata.com/` verwendet.

entstehen die Entitäten 2.3 und 2.5. Für alle Beispiele in diesem Kapitel werden Stichproben dieser Entitäten verwendet. Der vollständige Datensatz ist dem Datenträger hinterlegt (vgl. Anhang B).

ts	id	title	pid
1608114886	1	Ski	1
1584706430	2	Ski	2
1570291361	3	Tennis	3
1592145358	4	Puzzle	4
1621328950	5	Puzzle	5
1600291970	6	Tennis	6
1566420028	7	Fußball	7
1605128645	8	Tennis	8
1598257297	9	Fußball	9
1576953581	10	Tennis	10
1575469819	11	Puzzle	11
1599974175	12	Puzzle	12
1580844427	13	Puzzle	13
1596320758	14	Boxen	14
1590945526	15	Ski	15

Tabelle 2.2: Unverfremdete Mission-Entität

ts	id	title	pid
160811486	1	Ski	1
1584706403	2	Skz	2
1570291361	3	Tennis	3
1592153548	4	Puzzle	4
162q1328950	5	Puzzle	5
160029190	6	Teinns	6
566420028	7	Fußball	7
16g5128645	8	Tvennis	8
1598252797	9	Fußqball	9
1157569835	10	Tnnies	10
1575469z19	11	Puzzle	11
1599974175	12	Pzzle	12
1580l844427	13	Puzzle	13
m1596320758	14	oxen	14
159095526	15	bki	15

Tabelle 2.3: Verfremdete Mission-Entität

ts	id	name	score	address
1579119917	1	Grant Mclean	10	216 Per Straße
1588632462	2	Quinn Graves	10	Postfach 127, 5177 Suspendisse Ave
1572864968	3	Ahmed Whitley	10	Ap 973-7636 Facilisis St.
1579428798	4	Robert Wilder	5	Postfach 336, 7790 Sollicitudin Rd.
1608821565	5	Jin Ramos	5	809-1580 In Av.
1594284558	6	Gabriel Hays	5	Postfach 109, 5983 Leo, St.
1588900374	7	Amal Perkins	5	795-1438 Lorem St.
1577850872	8	Myles Pena	10	831-9275 Et Rd.
1619638047	9	Sawyer Evans	10	Ap 942-7326 Nullam St.
1590769665	10	Hunter Holland	100	5192 Arcu Straße
1562263887	11	Gareth Duran	100	Postfach 891, 2075 Orci. St.
1603914119	12	Kuame Potter	100	Postfach 791, 2049 Velit Straße
1572592670	13	Fuller Elliott	0	Postfach 205, 695 Hendrerit Av.
1600690001	14	Kaseem Yang	10	Ap 578-6320 Magna. Avenue
1601064496	15	Kieran York	10	2532 Lorem Rd.

Tabelle 2.4: Unverfremdete Player-Entität

ts	id	name	score	address
1579191917	1	Gmant Mclean	20	216 Per Straße
158863242	2	Qu Grannives	20	Pstfach 127 5177 Suspendisse Ave
157286468	3	Ahmed Whitley	20	Ap 973-7.tS sisilicaF 636
157942878	4	Robest Wilder	20	Postfach 336l 7790 Sollicitudin Rd.
1y08821565	5	Ji Ramos	20	800851-9 In Av.
1m94284558	6	Geirbal Hays	50	Pos39a15cLo 9ht 8f eo St.
588900374	7	Amal aPerkins	50	795-1438 eLorem St.
157850872	8	Myles Penw	0	831-9275 Et id.
16196t8047	9	Sawyer Evan	0	Ap 942-7326 Nullam St.
15907h69665	10	Huxter Holland	0	5192q Arcu Straße
15622e63887	11	Gareth Duraw	50	Postfach 890 2075 Orci. St.
16t03914119	12	Kuame Pottver	50	Postfach a 1e 4i rßl2tS9to97 Ve
1975657220	13	Fuller Elliott	100	Postfach 205 695 Hendrerit cAv.
1060690001	14	Kaseem Yan	100	Ap 578-620 Magna. Avenue
1601064t496	15	Kiera York	100	25q32 Lorem Rd.

Tabelle 2.5: Verfremdete Player-Entität

2.3 SCHEMA-EVOLUTIONS-PROZESS

Mithilfe eines Schema-Evolutions-Prozesses werden Veränderungen an Datenmodellen offengelegt. Datenmodellen widerfährt ein kontinuierlicher Veränderungsprozess, den Datenbankentwickler, zum Beispiel durch veränderte Anforderungen, verursachen [19][vgl. 2764]. Ein Datenmodell kann als Schnittstelle zwischen einem Datenproduzenten und einem Datenkonsumenten gesehen werden. Eine Veränderung des Datenmodells wird durch eine Schema-Veränderung in eine Datenbank eingebracht. Datenkonsumenten können Schema-Veränderungen nur indirekt identifizieren, weil eine Erweiterung eines Datenmodells für einen Datenkonsumenten zunächst nicht ersichtlich ist, wenn ein Datenzugriff weiterhin erfolgen kann. Für den Fall, dass die Datenschnittstelle zerbricht, liegt für den Datenkonsumenten die Vermutung nahe, dass eine Schema-Veränderung vollzogen wurde.

Eine Schema-Veränderung wird durch eine Schema-Operation eingeführt. Es können zwei Schema-Operationstypen unterschieden werden [19][vgl. 2767]:

- **Singletype-Schema-Operation:** Diese Schema-Operationen wirken sich auf genau eine Entität aus. Es ist möglich, über eine zeitliche Schemabetrachtung diese Operationen zu extrahieren. Sie umfassen die Operationen *Add*, *Remove* und *Rename*.
- **Multitype-Schema-Operation:** Es sind mehrere Entitäten an einer Schema-Operation beteiligt. Vertreter dieser Gruppe sind *Copy* und *Move*. Zusätzlich wird ein Verbund der Entitäten benötigt. Ein Verbund zweier Entitäten beschreibt, wie eine Entität zu einer anderen Entität in Relation gebracht werden kann.

Der Fokus dieser Arbeit liegt in der Erkennung der Multitype-Schema-Operationen *Copy* und *Move* (vgl. Kapitel 1). Diese können folgendermaßen unterschieden werden:

- Eine *Move*-Operation ist aufgetreten, wenn das Attribut einer Entität in einer vorherigen Schema-Version vorhanden ist und in einer nachfolgenden

Schema-Version fehlt. Einer weiteren Entität wird ein Attribut hinzugefügt. Dieses Attribut entstammt der ursprünglichen Entität.

- Eine *Copy*-Operation ist aufgetreten, wenn das Attribut einer Entität in einer vorherigen Schema-Version vorhanden ist und in einer darauf folgenden Schema-Version nicht entfernt wurde. Es ist aufwändig, diejenige Entität zu finden, deren Attribute in eine andere Entität kopiert wurden, da eine *Copy*-Operation keine schematischen Veränderungen an der ursprünglichen Entität verursacht. Falls ein Attribut in einer weiteren Entität durch eine Schema-Operation verfügbar gemacht wird, kann es sich um eine *Copy*-Operation handeln, wenn die ursprüngliche Entität des neu hinzugekommenen Attributs gefunden werden kann.

Gemäß der in [19][vgl. 2767] vorgestellten Domain-Specific-Language (DSL) können Multitype-Schema-Operationen beschrieben werden:

Listing 2.1: Gekürzte EBNF-Syntax zur Beschreibung von Multitype-Schema-Operationen aus [19][vgl. 2767]

```
multitypeop ::= move | copy;
move = "move" property "to" ( kind | property ) complexcond;
copy = "copy" property "to" ( kind | property ) complexcond;
complexcond = "where" joinconds ["and" conds];
joinconds = joincond {"and" joincond};
joincond = property "=" property;
conds = cond {"and" cond};
cond = property "=" value;
property = kind "." pname;
kind = [mname "."] kname;
kname = identifier;
mname = identifier;
pname = identifier;
```

Diese DSL wird in dieser Arbeit angewendet. Eine Schema-Veränderung impliziert eine neue Schema-Version. Eine Schema-Version ist die unveränderliche Darstellung eines Schemas zu einem definierten Zeitpunkt.

An welcher Stelle kann der Schema-Evolutions-Prozess ansetzen? Diese Frage ist bedeutend für die Anforderungen, die an einen solchen Prozess gestellt werden. Es können zwei Schema-Evolutions-Prozesse unterschieden werden:

- Im *beobachteten* Schema-Evolutions-Prozess liegt die Hoheit eines Datenmodells in den Händen des Datenbankentwicklers. Dieser hat die Möglichkeit, über eine Modifikation des eigenen Datenmodells einen Schema-Evolutions-Prozess anzustoßen. Da die Änderungen dem Datenbankentwickler und dem Anwendungsentwickler bekannt sind, können Schema-Veränderungen ohne die Berücksichtigung der Daten erfolgen. Sobald Daten den Einflussbereich der Datenbankentwicklung verlassen, lassen sich Schema-Veränderungen nur mit hohem Aufwand nachverfolgen. Dieser Ansatz wird von der Anwendung *Hackolade* implementiert [14].
- Dem gegenüber steht der *unbeobachtete* Schema-Evolutions-Prozess. Die Kontrolle über ein Datenmodell ist durch eine Datenweitergabe an Dritte durch einen Datenbankentwickler nicht mehr möglich. Es werden datengetriebene Verfahren gesucht, die Schema-Veränderungen automatisiert erkennen können. Alle Informationen werden einzig aus den Daten gezogen. In einer verteilten Softwarelandschaft ist dies von Vorteil, da unabhängig von den Datenlieferanten Schema-Veränderungen erkannt und protokolliert werden können.

Es wird ein unbeobachteter Schema-Evolutions-Prozess betrachtet, da in Kapitel 1 vorausgesetzt wird, dass keine Informationen ausgeführter Multitype-Schema-Operationen verfügbar sind. Alle Erkenntnisse sind durch datengetriebene Ansätze zu erheben.

In [18] wird ein Schema-Evolutions-Prozess vorgestellt, der durch eine alleinige Betrachtung von Schema-Informationen Multitype-Schema-Operationen extrahieren kann. Die folgenden Beispiele zeigen die Grenzen dieser Verfahren auf und motivieren die Notwendigkeit eines datengetriebenen Schema-Evolution-Prozesses.

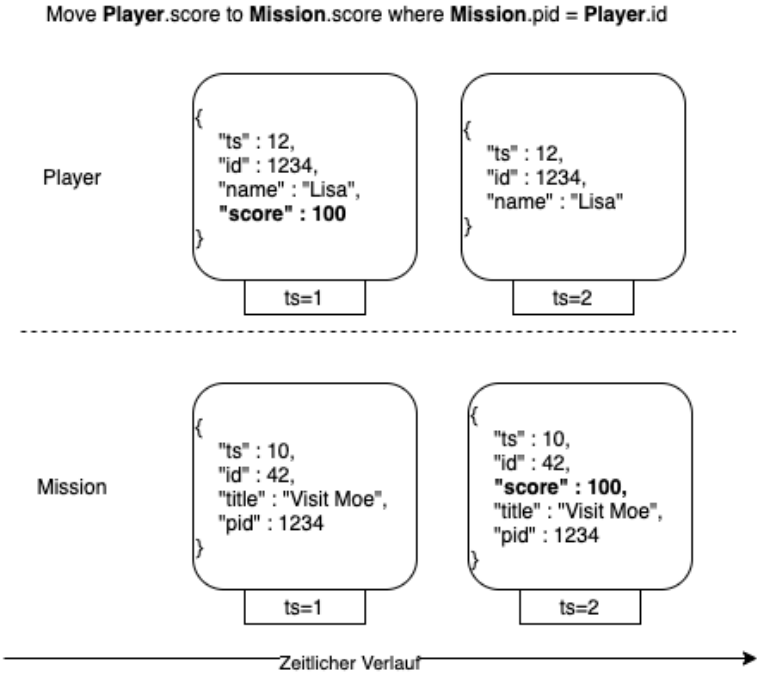


Abbildung 2.2: Move-Schema-Operation nach [24][vgl. 67]

In Abbildung 2.1 sind zwei Entitäten $\{Mission, Player\}$ zu sehen, die durch Primär- und Fremdschlüssel verbunden sind. Diese Informationen sind einer NoSQL-Datenbank nicht bekannt. Ein Datenbankentwickler muss eine Schema-Veränderung durchführen. Das Attribut *score* wird der *Player*-Entität entzogen und der *Mission*-Entität hinzugefügt. In Abbildung 2.2 ist die ausgeführte Schema-Operation zu sehen. Nachdem eine Schema-Veränderung erkannt wurde, kann anhand einer Schema-Analyse nachvollzogen werden, dass das Attribut *score* von der *Player*-Entität in die *Mission*-Entität verschoben wurde. Es liegt die Multitype-Schema-Operation *Move* vor. Somit wird ein Verbund der Entitäten benötigt. Die Primär- und Fremdschlüssel-Beziehungen der Entitäten können durch eine weitere Schema-Analyse erhoben werden. Der Autor [17] schlägt vor die Attributsnamen auf mögliche Primär- und Fremdschlüssel-Beziehungen zu untersuchen.

Die bisherige Vorgehensweise muss erweitert werden. Dies zeigt das folgende Beispiel:

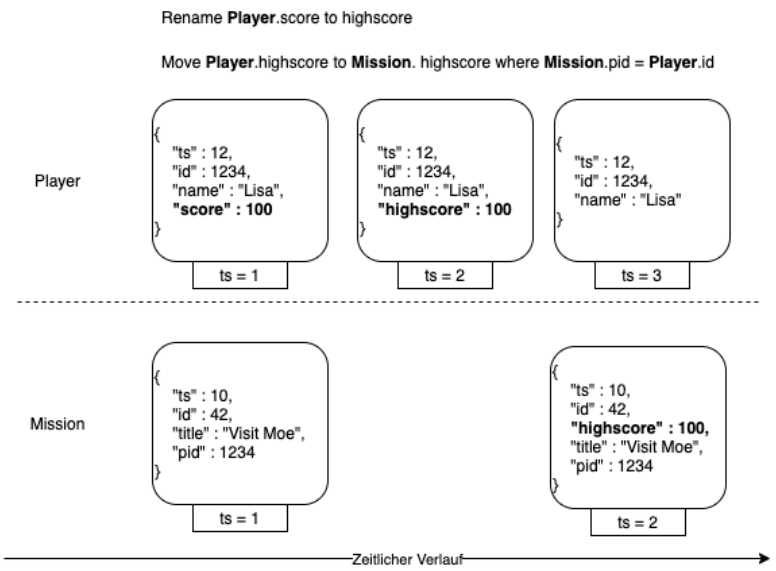


Abbildung 2.3: Kaskadierte Move-Schema-Operation nach [24][vgl. 67]

In Abbildung 2.3 werden die bekannten Entitäten *Player* und *Mission* wie in Abbildung 2.2 verwendet. Eine Veränderung des Schemas wird im Timestamp[7] $ts = 2$ erkannt. Das Attribut *score* wird der *Player*-Entität entnommen und der *Mission*-Entität mit dem geänderten Namen *highscore* hinzugefügt. Durch eine alleinige Betrachtung der Schema-Veränderungen kann nicht entschieden werden, welche Schema-Operation durchgeführt wurde. Die Attribute *score* und *highscore* sind lexikalisch verschieden. Die Schema-Veränderungen können nicht zueinander in Beziehung besetzt werden. Es entstehen zwei Schema-Operationen: Eine Schema-Operation für das Entfernen des Attributs *score* aus der *Player*-Entität und eine Schema-Operation für das Hinzufügen des Attributs *highscore* zu der *Mission*-Entität. Mithilfe eines datengetrieben Schema-Evolutions-Prozess werden auch Multitype-Schema-Operationen identifiziert, dessen Schemas verschieden sind. Im weiteren Verlauf dieser Arbeit steht ein datengetriebene Schema-Evolutions-Prozess im Vordergrund. Die Verfahren Domänenklassifikation, Ähnlichkeits- und Verbundberechnung sind die Kernkomponenten des Schema-Evolutions-Prozesses und werden in den folgenden Abschnitten eingeführt.

2.4 DOMÄNENKLASSEN

Eine Domäne $dom(X)$ eines Attributs X ist eine Abstraktion der Attributswerte in vordefinierten Klassen und ist mit dem Datentyp eines Attributs vergleichbar. Im Gegensatz zum Datentyp ist die Domäne nicht abhängig von einer konkreten Implementierung durch einen Datenbankhersteller, sondern ist an die mathematischen Skalenniveaus gekoppelt [8][vgl. 2]. Zusätzlich muss durch das Konzept Schema-On-Read ein definierter Datentyp nicht auch zu den hinterlegten Daten passen [34][vgl. 2]. So kann ein numerisches Attribut auch in einem alphabetischen Datentyp gespeichert werden. In einem solchen Fall kann der Datentyp keine Aussage über die Daten geben. Für NoSQL-Datenbanken ist ein Schema durch die Domänenklassen der Attribute einer Entität bestimmt. Dies ermöglicht die Integrierung beliebiger NoSQL-Datenbanken in den Schema-Evolutions-Prozess, da das Schema

7 Unterschiedliche Schema-Versionen können zum Beispiel durch Zeitmarker unterschieden werden.

kein datenbankabhängiges Konstrukt ist. Alle Erkenntnisse sind aus den Daten zu gewinnen. Es können nach [8] folgende Domänenklassen beschrieben werden:

- Ein Attribut wird der Domänenklasse **Ordinal** zugeordnet, wenn diskrete Werte vertreten sind, eine sinnvolle Rangordnung gebildet werden kann und die Abstände zwischen den Rängen nicht interpretierbar sind. Zum Beispiel lassen sich Schulnoten und Steuerklassen in eine Reihenfolge bringen. Die Abstände zwischen den verschiedenen Rängen sind nicht interpretierbar.
- **Metrische** Attribute sind stetig und können in eine sinnvolle Rangordnung gebracht werden. Zusätzlich sind die Abstände zwischen den Rängen interpretierbar. Beispiel: Distanzen und (Geld)Beträge.
- Die Domänenklasse **Nominal** beschreibt diskrete Werte, deren Abstände keine sinnvolle Rangordnung zulassen. Vertreter sind zum Beispiel Geschlecht, Familienstand oder Herkunft. Die Domänenklasse **Nominal** lässt sich weiter unterteilen:
  - **Kategoriale** Domänenklassen haben die Einschränkung, dass die Anzahl der einmaligen Attributsausprägungen endlich ist.
  - In der Domänenklasse **Dichotom** ist die Anzahl der einmaligen Attributsausprägungen auf zwei begrenzt.
- Die **alphabetische** Domänenklasse umfasst Attributswerte, die sich in der Menge $[A - Za - z]$ befinden.

Zusätzlich werden drei weitere Domänen eingeführt, die den Schema-Evolutions-Prozess unterstützen werden:

- Die Domänenklasse **Unique-Column-Combination (UCC)** setzt die Einmaligkeit aller Attributswerte voraus. Diese Eigenschaft ist eine Grundvoraussetzung für einen Primärschlüssel [17, vgl. 444] .
- Die **Date**-Domäne beschreibt einen Zeit- oder Datumswert.
- Mit der **None**-Domäne werden diejenigen Attribute bezeichnet, für die keine Domänenklassen gefunden werden können. Meist ist dies der Fall, wenn keine Attributswerte vorhanden sind.

In Abbildung 2.4 ist die Entscheidungsgrundlage für die Domänenklassifikation eines Attributs gezeigt:

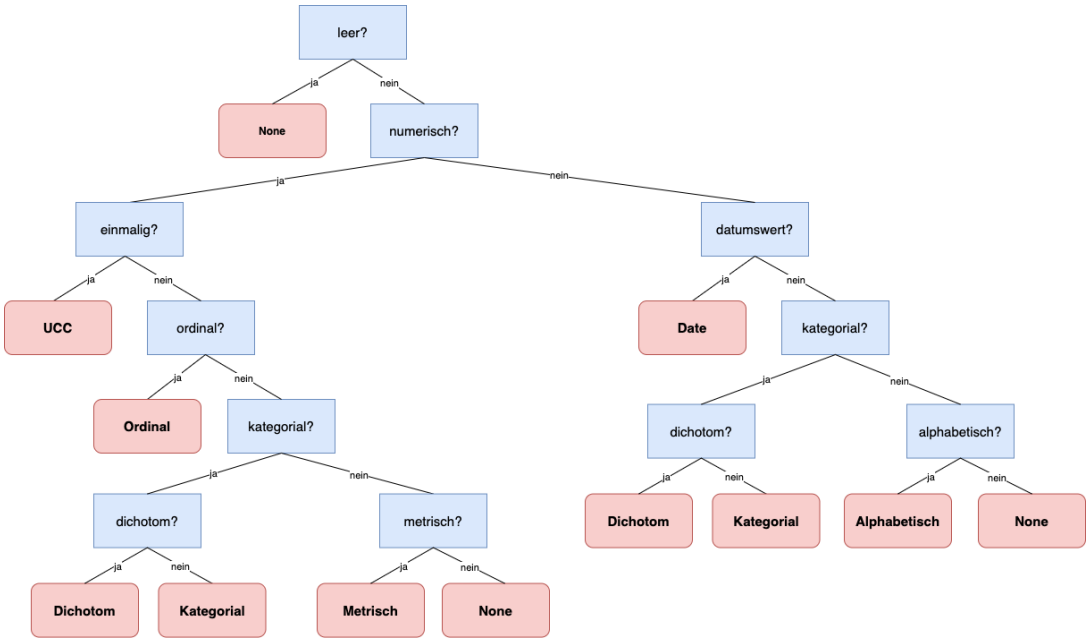


Abbildung 2.4: Domänenklassifikation

Die Entscheidungsknoten der Abbildung 2.4 sind in der folgenden Tabelle beschrieben:

Domäne	**Entscheidungsknoten**	**Entscheidungsgrundlage**
None	leer?	Es sind weniger als 30 nicht leere Datensätze in einem Attribut vorhanden.
-	numerisch	Ein Attribut ist numerisch, wenn dieses Zahlen enthält.
UCC	einmalig?	Die Anzahl der einmaligen Ausprägungen ist im Vergleich zur Gesamtanzahl aller Ausprägungen größer als θ[1].
Date	datumswert?	Datum/Zeitwerte liegen vor. Überprüfung anhand eines Patterns (z.B. *YYYY-MM-DD*).
Ordinal	ordinal?	Ein Attribut kann diskrete Werte aufweisen.
Kategorial	kategorial?	Die Anzahl der einmaligen Ausprägungen ist definierbar endlich.
Dichotom	dichotom?	Es liegen maximal zwei Attributausprägungen vor.
Alphabetisch	alphabetisch?	Ein Attribut weist alphabetische Attributswerte auf.
Metrisch	metrisch?	Stetige numerische Attributswerte können verzeichnet werden.

Tabelle 2.6: Detaillierte Domänenklassifikation

[1] In Abhängigkeit der benötigten Sensitivität sind übliche Werte für θ: $0.8, 0.85, 0.9, \ldots$

2.4.1 *Anwendung*

Für das *Mission-Player*-Datenmodell könne folgende Domänenklassen gemäß der Domänenklassifikation aus Abbildung 2.4 ermittelt werden:

Entität	**Attribut**	**Domäne**
Mission	id	UCC
Mission	ts	Ordinal
Mission	title	Kategorial
Mission	pid	Ordinal
Player	score	Metrisch
Player	id	UCC
Player	ts	Ordinal
Player	name	Alphabetisch
Player	address	Alphabetisch

Tabelle 2.7: Domänenklassen des Mission-Player-Datenmodells

Die Domänenklassifikation ist ein notwendiger Schritt im Schema-Evolution-Prozess, da eine Schema-Veränderung nur erkannt werden kann, wenn ein Schema definiert ist. Für neue Datenquellen muss der Prozess der Domänenklassifikation einmalig durchgeführt werden. Im weiteren Schema-Evolutions-Prozess müssen dieje-

nigen Attribute identifiziert werden, denen eine Schema-Veränderung widerfahren ist. Dies wird durch eine Analyse der Schema-Versionen erreicht.

2.5 ÄHNLICHKEITSBERECHNUNG

Mithilfe eines Ähnlichkeitsmaßes werden Attribute durch das Teilen einer gemeinsamen Semantik zusammengeführt.

2.5.1 *Funktionseigenschaften*

Für die Berechnung von Ähnlichkeiten muss der Begriff der Ähnlichkeit abgegrenzt werden. Für eine Ähnlichkeitsberechnung werden zwei Vektoren $\{X, Y\}$ und ein Ähnlichkeitsmaß $\sim$ benötigt. Hierbei ist $\sim$ eine abstrakte Funktion. Ein Ähnlichkeitsmaß muss die folgenden Funktionseigenschaften erfüllen [1][vgl. 17ff]. Diese sind:

$$\textit{Kommutativität} : X \sim Y = Y \sim X$$

Mithilfe der **Kommutativität** hat die Reihenfolge der Funktionsargumente keinen Einfluss auf das Ergebnis [1][vgl. 17]. Dies hat den positiven Effekt, dass der Suchraum um die Hälfte reduziert werden kann [27][vgl. 94].

$$\textit{Transitivität} : (X \sim Y)\ \textit{und}\ (Y \sim Z) \Rightarrow X \sim Z$$

Wenn zwischen den Vektoren X und Y eine Ähnlichkeit besteht und auch zwischen den Vektoren Y und Z, kann durch das Anwenden der **Transitivität** eine Ähnlichkeit zwischen X und Z nachgewiesen werden [27][vgl. 94]. Dies gilt nur, wenn es Schnittmengen zwischen den Mengen $X \sim Y$ und $Y \sim Z$ gibt. Je höher der Ähnlichkeitswert zwischen den Vektoren ist, desto wahrscheinlicher wird es, dass eine Schnittmenge existiert und die Eigenschaft der Transitivität erfüllt werden kann. Das folgende Beispiel zeigt dies:

- $\{X, Y, Z\}$ beinhalten jeweils eine Menge von numerischen Werten.
- Es wurde über ein Ähnlichkeitsmaß eine Beziehung zwischen X und Y ($X \sim Y$) und zwischen Y und Z ($Y \sim Z$) gefunden.
- Nach der Anwendung der Transitivitätseigenschaft muss $X \sim Y\ \textit{und}\ Y \sim Z \Rightarrow X \sim Z$ gelten.
- Die Evidenz der Ähnlichkeiten definiert nun, ob die $X \sim Z$ gehalten werden kann: Angenommen die Ähnlichkeit zwischen $X \sim Y$ ist nicht signifikant, das heißt, jene Beziehung kann nicht bijektiv[8] sein, da Elemente aus X nicht auf exakt **ein** Element in der Zielmenge von Y zugeordnet werden können. Somit existieren in X und Y Werte, die **nicht** zueinander ähnlich sind. Diese Überlegung kann auch zwischen $Y \sim Z$ angewendet werden. Die Menge an Werten, die nicht zugeordnet werden mit $\alpha = X \nsim Y$ und $\gamma = Y \nsim Z$ definiert. α und γ können disjunkt[9] sein, wenn die Menge α für $Y \sim Z$ und γ für $X \sim Y$ verwendet wird. Durch diese Disjunktion kann nun nicht $X \sim Z$ geschlossen werden. Das bedeutet, je höher die Ähnlichkeiten zwischen den Vektoren ist, desto kleiner werden α und γ und damit steigt die Wahrscheinlichkeit, dass die Mengen nicht disjunkt sind und eine Ähnlichkeit zwischen X und Z kann nachgewiesen werden.

In Abbildung 2.5 sind drei Mengen (X, Y, Z) zu sehen die zueinander Ähnlichkeiten aufweisen ($X \sim Y$ und $Y \sim Z$). Es kann in der zweiten Abbildung die Eigenschaft

8 Jedes Element aus X findet genau ein Partnerelement in Y.
9 Die Vektoren X und Y haben keine gemeinsame Schnittmenge.

der Transitivität nicht gehalten werden, da zwischen X und Z keine Schnittmenge besteht.

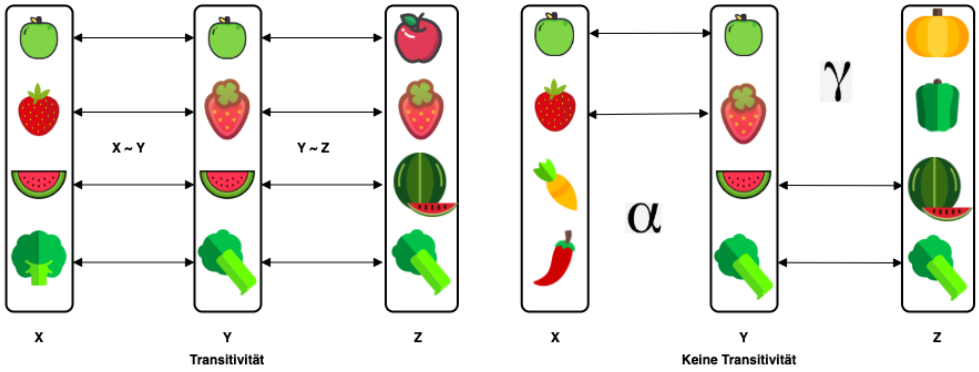


Abbildung 2.5: Grenzen der Transitivität

Durch das Anwenden der **Transitivität** kann der Suchraum weiter reduziert werden, da die Kombination $X \sim Z$ nicht überprüft werden muss.

$$\textit{Eindeutigkeit} : (X \sim Y)^* = (X \sim Y)$$

Ein mehrmaliges Ausführen eines Ähnlichkeitsmaßes auf immer gleiche Vektoren muss durch die Funktionseigenschaft der **Eindeutigkeit** immer ein identisches Ergebnis liefern [27][vgl. 94]. Ein Determinismus der Funktion wird erwartet.

$$\textit{Skalierbarkeit} : s(X) \sim s(Y) = X \sim Y$$

Eine Massendatenverarbeitung kann nur auf Stichproben erfolgen [13][vgl. 166]. Diese wird durch die Eigenschaft der **Skalierbarkeit** erfüllt. Eine Stichprobe eines Vektors X wird mit $s(X)$ definiert. Ein berechneter Ähnlichkeitswert auf einer Stichprobe muss identisch mit einer Ähnlichkeitsberechnung auf dem vollständigen Datensatz sein [20][vgl. 2]. Es existieren mehrere Verfahren, wie Stichproben generiert werden können:

- **Zufallsstichprobe**: Es werden aus einem Datensatz zufällige Werte gezogen und diese der Stichprobe hinzugefügt. Solche Stichproben variieren bei mehrfacher Ausführung[10]. Es kann nicht bestimmt werden, ob die Stichprobe die ursprüngliche Attributwertverteilung repräsentativ annähern kann, sodass eine Ausführung auf dem gesamten Datensatz eine Abweichung des Ähnlichkeitswertes verursachen könnte [20][vgl. 2].
- **Prozentual**: Es wird ein vordefinierter Prozentwert an Daten aus einem Datensatz entnommen. Auch in diesem Verfahren kann nicht sichergestellt werden, dass eine repräsentative Stichprobe erzeugt wird [20][vgl. 3].
- **Bucketing**: Mithilfe einer Hash-Funktion wird ein Datensatz in Buckets unterteilt. Nun kann bestimmt werden, wie viele Werte aus einem Bucket für eine Stichprobe verwendet werden sollen. Da ein Bucket durch das Anwenden einer Hash-Funktion ähnliche und identische Attributswerte bündelt, müssen nur wenige Elemente aus einem Bucket für eine Stichprobe verwendet werden. In einer zweiten Stichprobe für den Datensatz Y wird die identische Hash-Funktion angewendet. So werden die Datensätze diskretisiert. Da kein

10 Ausgenommen, es wird ein **seed**-Wert für die Initialisierung des Zufallszahlengenerators definiert.

Zufall in der Bestimmung der Stichprobe verwendet wird, können mithilfe des Bucketings die Eigenschaften der **Eindeutigkeit** und der **Skalierbarkeit** erfüllt werden [20][vgl. 3].

$$Idempotenz : X \sim X = X = 1$$

Selbige Attribute $\{X, X\}$ müssen durch die Funktionseigenschaft der **Idempotenz** nicht berücksichtigt werden [27][vgl. 94].

$$Domänenkompatibilität : dom(X)\ comp\ dom(Y)|X \sim Y$$

Mit der Funktionseigenschaft der **Domänenkompatibilität** wird überprüft, ob es ein Ähnlichkeitsmaß gibt, die für jene Attributswerte angewendet werden kann. Die Funktion *comp* betrachtet die Domänen beider zu vergleichenden Attribute und ermittelt basierend auf einer Domänenkompatibilitätsmatrix ein passendes Ähnlichkeitsmaß [8][vgl. 7].

$$Interpretierbarkeit : X \sim Y \in [0, 1]$$

Ein Ähnlichkeitswert muss für die Erfüllung der Funktionseigenschaft **Interpretierbarkeit** zwischen 0 und 1 (0% und 100%) liegen [5, vgl. 7].

$$Anwendungsbereich : e_1.X \neq e_2.Y \begin{cases} true, & X \sim Y \\ false, & - \end{cases}$$

Der **Anwendungsbereich** definiert, welche Entitäten $\{e_1, e_2\}$ für eine Ähnlichkeitsberechnung betrachtet werden müssen. Nur unterschiedliche Entitäten werden für eine Ähnlichkeitsberechnung zugelassen. Somit werden Ähnlichkeiten innerhalb einer Entität nicht betrachtet [8][vgl. 7].

Es wird ein Parameter θ eingeführt. Dieser beschreibt die untere Schranke ab welchem Ähnlichkeitswert zwei Datensätze als *ähnlich* oder als *disjunkt* betrachtet werden [8][vgl. 7]:

$$X \sim Y \geq \theta = \begin{cases} true, & \text{ähnlich} \\ false, & \text{disjunkt} \end{cases}$$

Zusammenfassend werden folgende Funktionseigenschaften an ein Ähnlichkeitsmaß gestellt:

- **Kommutativität**: $X \sim Y = Y \sim X$
- **Transitivität**: $(X \sim Y)\ und\ (Y \sim Z) \Rightarrow X \sim Z$
- **Eindeutigkeit**: $(X \sim Y)^* = (X \sim Y)$
- **Skalierbarkeit**: $s(X) \sim s(Y) = X \sim Y$
- **Idempotenz**: $X \sim X = X = 1$
- **Interpretierbarkeit**: $X \sim Y \in [0, 1]$
- **Domänenkompatibilität**: $dom(X)\ comp\ dom(Y)|X \sim Y$

2.6 VERBUNDBERECHNUNG

Mithilfe einer Verbundberechnung werden zwei Entitäten in eine semantische Beziehung gebracht.

2.6.1 *Problemdefinition*

Das Konstrukt aus Primär- und Fremdschlüssel-Beziehung kommt aus der relationalen Datenmodellierung. Eine Menge an Relationen können durch ausgezeichnete Schlüssel miteinander verbunden werden. In NoSQL-Datenbanken fehlen solche Auszeichnungen meist [23][vgl. 116]. Im Schema-Evolutions-Prozess sind Primär- und Fremdschlüssel-Beziehungen für die Erkennung von Multitype-Schema-Operationen wichtig.

2.6.2 *Funktionseigenschaften*

Eine Verbundberechnung kann durch die abstrakte Funktion $\bowtie$ beschrieben werden (vgl. Abschnitt 2.5). Realisierungen dieser Funktion müssen die folgenden Funktionseigenschaften aufweisen:

- **Kommutativität**: $X \bowtie Y = Y \bowtie X$ [27][vgl. 94]
- **Transitivität**: $(X \bowtie Y)\ und\ (Y \bowtie Z) \Rightarrow X \bowtie Z$ [27][vgl. 94]
- **Eindeutigkeit**: $(X \bowtie Y)^* = (X \bowtie Y)$ [20][vgl. 3]
- **Skalierbarkeit**: $s(X) \bowtie s(Y) = X \bowtie Y$ [20][vgl. 3]
- **Idempotenz**: $X \bowtie X = X = 1$ [27][vgl. 94]
- **Interpretierbarkeit**: $X \bowtie Y \in [0, 1]$ [5, vgl. 7]
- **Domänenkompatibilität**: $dom(X)\ comp\ dom(Y) | X \bowtie Y$ [8][vgl. 7].

Es wird ein Parameter θ eingeführt,. Dieser beschreibt eine untere Schranke ab welchem Verbundwert zwei Datensätze als *verbunden* beziehungsweise als *disjunkt* betrachtet werden können:

$$X \bowtie Y \geq \theta = \begin{cases} true, & verbunden \\ false, & disjunkt \end{cases}$$

3 KONZEPTION

Im Grundlagenkapitel sind Verfahren und Begrifflichkeiten definiert worden. Diese werden im Folgenden für die Identifizierung von Multitype-Schema-Operationen in einem Schema-Evolutions-Prozess kombiniert. Für die Ähnlichkeitsberechnung wird eine Domänenkompatibilitätsmatrix aufgebaut. Mit dieser ist es möglich, die Anzahl an Kandidaten für die Ähnlichkeitsberechnung zu reduzieren. Selbige Überlegung wird für die Verbundberechnung durchgeführt. Der Schema-Evolutions-Prozess wird anschließend architektonisch aufgebaut. Abschließend wird die Frage nach der Rechtmäßigkeit der vorgestellten Verfahren geklärt.

3.1 VORBEREITUNG

Im Folgenden werden anhand des bekannten *Mission-Player*-Datenmodells die einzelnen notwendigen Schritte zur Identifizierung von Multitype-Schema-Operationen verdeutlicht. Es liegen folgende Datensätze in der Datenbank *data* vor:

ID	**ts**	**name**	**score**	**address**
1	1579119917	Grant Mclean	31	216 Per Straße
2	1588632462	Quinn Graves	22	Postfach 127, 5177 Suspendisse Ave
3	1572864968	Ahmed Whitley	42	Ap 973-7636 Facilisis St.
4	1579428798	Robert Wilder	84	Postfach 336, 7790 Sollicitudin Rd.
5	1608821565	Jin Ramos	55	809-1580 In Av.
6	1594284558	Gabriel Hays	61	Postfach 109, 5983 Leo, St.
7	1588900374	Amal Perkins	6	795-1438 Lorem St.
8	1577850872	Myles Pena	7	831-9275 Et Rd.
9	1619638047	Sawyer Evans	69	Ap 942-7326 Nullam St.
10	1590769665	Hunter Holland	81	5192 Arcu Straße

Tabelle 3.1: Player-Entität in der Schema-Version v_0

ID	ts	name	address
1	1579119917	Grant Mclean	216 Per Straße
2	1588632462	Quinn Graves	Postfach 127, 5177 Suspendisse Ave
3	1572864968	Ahmed Whitley	Ap 973-7636 Facilisis St.
4	1579428798	Robert Wilder	Postfach 336, 7790 Sollicitudin Rd.
5	1608821565	Jin Ramos	809-1580 In Av.
6	1594284558	Gabriel Hays	Postfach 109, 5983 Leo, St.
7	1588900374	Amal Perkins	795-1438 Lorem St.
8	1577850872	Myles Pena	31-9275 Et Rd.
9	1619638047	Sawyer Evans	Ap 42-7326 Nullam St.
10	1590769665	Hunter Holland	5192 Arcu Straße

Tabelle 3.2: Player-Entität in der Schema-Version v_1

ID	ts	title	pid
1	1608114886	Ski	1
2	1584706430	Ski	2
3	1570291361	Tennis	3
4	1592145358	Puzzle	4
5	1621328950	Puzzle	5
6	1600291970	Tennis	6
7	1566420028	Fußball	7
8	1605128645	Tennis	8
9	1598257297	Fußball	9
10	1576953581	Tennis	10

Tabelle 3.3: Mission-Entität in der Schema-Version v_0

ID	ts	title	pid	score
1	1608114886	Ski	1	31
2	1584706430	Ski	2	22
3	1570291361	Tennis	3	42
4	1592145358	Puzzle	4	84
5	1621328950	Puzzle	5	55
6	1600291970	Tennis	6	61
7	1566420028	Fußball	7	6
8	1605128645	Tennis	8	7
9	1598257297	Fußball	9	69
10	1576953581	Tennis	10	81

Tabelle 3.4: Mission-Entität in der Schema-Version v_1

Es wird eine Multitype-Schema-Operation durchgeführt. Das Attribut *score* wird der *Player*-Entität entzogen und der *Mission*-Entität hinzugefügt.

3.2 SCHEMA-EVOLUTIONS-ARCHITEKTUR

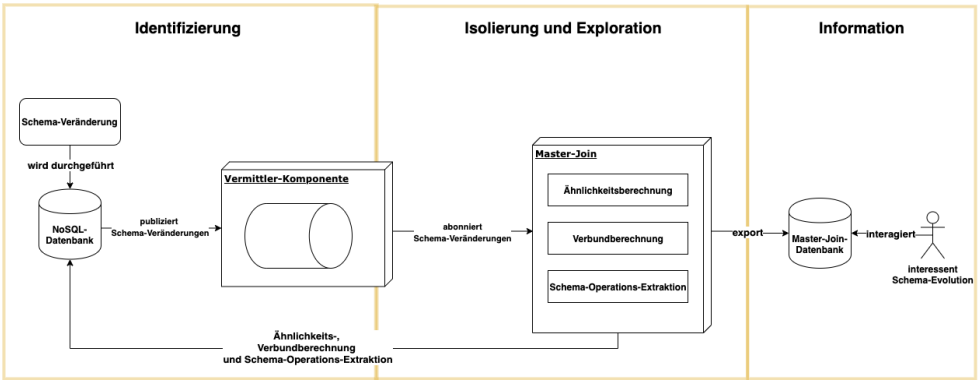


Abbildung 3.1: Konzeptionelle Schema-Evolutions-Architektur

In Abbildung 4.2 ist die Schema-Evolutions-Architektur für die Identifizierung von Multitype-Schema-Operationen zu sehen. Die Architektur lässt sich in vier Ebenen unterteilen:

1. Identifizierung
2. Isolierung
3. Exploration
4. Information

3.2.1 *Identifizierung*

Eine Schema-Veränderung initiiert den Schema-Evolutions-Prozess. Sobald eine Schema-Veränderung erkannt wurde, wird ein Datagramm erzeugt. Ein Datagramm beschreibt die erkannte Schema-Veränderung und besteht aus den Informationen *Datenbank* und *Entität*, die jeweils durch einen Punkt getrennt sind. Das Datagramm wird einer Vermittler-Komponente übergeben und dieser Vermittler sorgt für die garantierte und einmalige Auslieferung der Datagramme an die Komponente *Master-Join*. Der Master-Join ist die Kernkomponente zur Identifizierung von Multitype-Schema-Operationen. Dieser umfasst die Ähnlichkeits- und Verbundberechnung, die Extraktion der Schema-Operationen sowie die Speicherung aller Erkenntnisse in einer weiteren Datenbank.

3.2.2 *Isolierung*

Der Schema-Evolutions-Prozess beginnt mit dem Empfang eines Datagramms durch die Vermittler-Komponente. Nach dem Interpretieren des Datagramms werden bestehende Schema-Informationen aus der Master-Join-Datenbank geladen. Anschließend wird ermittelt, welche Attribute verändert wurden. Dieses Attribut kann durch einen Abgleich mit vorherigen Schema-Versionen aus der Master-Join-Datenbank ermittelt werden. Dies ist möglich, da zur Initialisierung des Master-Joins alle Schema-Informationen einmalig vollständig extrahiert und gespeichert worden waren. Ein Schema wird durch eine Datenanalyse für jede Entität erzeugt. Das Schema einer Entität bündelt Attribute und ein Attribut besteht aus einem Namen sowie einer Domänenklasse. Diese Informationen können datengetrieben erhoben werden.

Wenn ein mengenwertiges Attribut erkannt wird, muss dieses für den weiteren Schema-Evolutions-Prozess vorverarbeitet werden, da die Ähnlichkeits- und Verbundmaße atomare Attributswerte voraussetzen. Atomare Attributswerte werden

durch einen Linearisierungsprozess aus einem mengenwertigen Attribut erzeugt. Die folgenden mengenwertigen Modelle werden nach [40] unterschieden: Array, Map und Struct, Kombinationen mengenwertiger Modelle für ein Attribut sind möglich. Verfahren für das Linearisieren der mengenwertigen Attribute lassen sich in [40] und [21] finden. Diese werden nicht beschrieben, da der Fokus dieser Arbeit im Schema-Evolutions-Prozess liegt. Das Linearisieren der mengenwertigen Attribute ist ein Vorverarbeitungsschritt und in der Konzeption und der prototypischen Entwicklung der Verfahren werden ausschließlich atomare Attribute berücksichtigt.

Anschließend können Domänenklassen generiert werden. Die Möglichkeit, Schemas für Entitäten zu definieren, wird nicht berücksichtigt, da auch NoSQL-Datenbanken ohne diese Möglichkeit betrachtet werden müssen. Durch diese Abstraktion wird eine unabhängige Untersuchung unterschiedlicher NoSQL-Datenbanktypen möglich. Die Verfahren sind nicht proprietär auf ein DBMS zugeschnitten. Die Domänenklassifikation ist unabhängig von einer konkreten NoSQL-Datenbank.

3.2.3 *Exploration*

Wenn eine Schema-Veränderung erkannt wurde und die beobachtete Entität durch einen Abgleich mit bestehenden Schema-Informationen isoliert wurde, muss die Schema-Historie betrachtet werden. Eine Entität kann in mehreren Schema-Versionen vorliegen. Für das Bilden von Schema-Versionen können zwei Verfahren unterschieden werden:

- Eine Schema-Veränderung wird durch das Erzeugen weiterer Entitäten realisiert. Mithilfe einer Basisentität kann für eine konkrete Entität die Schema-Historie nachvollzogen werden. Die Basisentität ist eine schema-versions-unabhängige Darstellung einer Entität. Hierfür muss ermittelt werden, nach welchem Pattern eine Schema-Version gebildet wird[1]. Zum Beispiel kann einer Entität ein numerisches Zeichen an den Entitätsnamen angefügt werden[2]. Ein Datenzugriff kann auf den verschiedenen Schema-Versionen einer Entität erfolgen. Zusätzlich besteht die Möglichkeit, über ausgewiesene Datensichten einen konsolidierten Datenbestand aller Schema-Versionen zu definieren [39][vgl. 1f].
- Eine Schema-Veränderung führt zu einer Veränderung innerhalb einer Entität. Ein Datenzugriff erfolgt ausschließlich auf der aktuellen Schema-Version einer Entität. Die Wiederherstellung einer vorherigen Schema-Version ist durch eine weitere Schema-Veränderung jener Entität möglich.

Es wird das Verfahren aus [39] betrachtet, da jede Schema-Veränderung eine unveränderliche Operation sein sollte. Diese Unveränderlichkeit bedeutet, dass ein Zugriff auf verschiedenen Schema-Versionen möglich ist. Jede Schema-Version ist in der Datenbank gespeichert und zugreifbar. Dem entgegen steht die Modifizierung einer Entität. Innerhalb einer Entität können mehrere Schema-Versionen vorhanden sein. Für den Zugriff auf eine vorangegangene Schema-Version müssen die Schema-Versionen untereinander abgrenzbar sein. Dieser Aufwand wird vermieden, wenn für jede Schema-Version neue Entitäten eingeführt werden.

Der Schema-Evolutions-Prozess wird weitergeführt. Durch die Identifizierung von Basisentitäten kann eine Schema-Historie der Entitäten gebildet werden. Für den Schema-Evolutions-Prozess ist die Differenz zwischen der aktuellen und der vorherigen Schema-Version einer Entität zu ermitteln. Für jedes Attribut der Diffe-

1 Das Pattern ist dem Master-Join über einen Parameter mitzuteilen.
2 Für die *Mission*-Entitäten können Schema-Versionen mit $Mission\_1, Mission\_2, \ldots, Mission\_n$ definiert werden.

renzmenge wird geprüft, ob dieses Attribut in der aktuellen Schema-Version hinzugefügt oder entfernt wurde. Dies ist für die Unterscheidung zwischen den Multitype-Schema-Operationen *Move* und *Copy* notwendig.

Jedes Attribut muss mit allen anderen Attributen auf eine Ähnlichkeitsbeziehung überprüft werden. Die Funktionseigenschaften aus Kapitel 2.5 können die Anzahl der möglichen zu überprüfenden Attributspaare reduzieren. Die Idee ist, dass eine neue Ähnlichkeitsbeziehung in den Daten erzeugt wird, wenn eine Multitype-Schema-Operation durchgeführt wurde. Für die Berechnung der Ähnlichkeit zwischen zwei Attributen wird mithilfe der Domänenkompatibilitätsmatrix ein passendes Ähnlichkeitsmaß gefunden.

Wird für ein Attribut eine Ähnlichkeitsbeziehung nachgewiesen, muss ein Entitätsverbund berechnet werden. Ein Entitätsverbund beschreibt eine semantische Beziehung zwischen zwei Entitäten und kann durch Primär- und Fremdschlüssel-Beziehungen ausgedrückt werden. Da auch für einen Entitätsverbund alle Attributskombinationen zwischen den betroffenen Entitäten berechnet werden müssen, werden die Funktionseigenschaften aus Kapitel 2.6 angewendet. Zusätzlich kann durch eine Betrachtung von Statistiken die Menge der zu überprüfenden Attributspaare reduziert werden.

Für eine *Move*-Schema-Operation muss in einer vorherigen Schema-Version einer Entität ($e^a_{v_0}$) ein Attribut (a) vorhanden sein, welches in einer Folgeversion desselben Schemas ($e^a_{v_1}$) entfällt. Zusätzlich müssen Attribute ($e^b_{v_1}.b$) gefunden werden, die dem Attribut ($e^a_{v_0}.a$) ähnlich sind. In einem solchen Fall, wird ein Entitätsverbund berechnet ($e^a_{v_1}.id = e^b_{v_1}.fid$)[3]. Mithilfe dieser Information kann die ursprüngliche Multitype-Schema-Operation ermittelt werden:

$$MOVE\ e^a_{v_0}.a\ to\ e^b_{v_1}.b\ WHERE\ e^a_{v_1}.id\ =\ e^b_{v_1}.fid$$

Für eine *Copy*-Schema-Operation muss ein Attribut (a) in einer vorherigen Schema-Version ($e^a_{v_0}$) vorhanden sein, welches in einer Folgeversion desselben Schemas ($e^a_{v_1}$) weiterhin besteht. Der Schema-Evolutions-Prozess wird nur für diejenigen Entitäten ausgeführt, denen durch eine Schema-Operation eine Veränderung widerfahren ist. Analog der *Move*-Schema-Operation werden nach Bekanntwerden einer Schema-Operation alle ähnlichen Attribute gesucht. Zwischen Entitäten, die ähnliche Attribute teilen, wird ein Verbund berechnet ($e^a_{v_0}.id = e^b_{v_1}.fid$). Abschließend kann eine *Copy*-Schema-Operation zusammengefasst werden:

$$COPY\ e^a_{v_0}.a\ to\ e^b_{v_1}.b\ WHERE\ e^a_{v_0}.id\ =\ e^b_{v_1}.fid$$

3.2.4 *Information*

Die ermittelten Multitype-Schema-Operationen werden den Datenkonsumenten zur Verfügung gestellt. Der Schema-Evolutions-Prozess ist beendet.

MASTER-JOIN

Die Kernkomponente für die Identifizierung von Multitype-Schema-Operationen ist der Master-Join. Die folgenden Darstellung zeigt die Aufgaben des Master-Joins.

3 Die Schreibweise $e^a_{v_1}.id$ besagt, dass die Entität e^a in der Schema-Version v_1 ein Attribut mit dem Namen *id* besitzt.

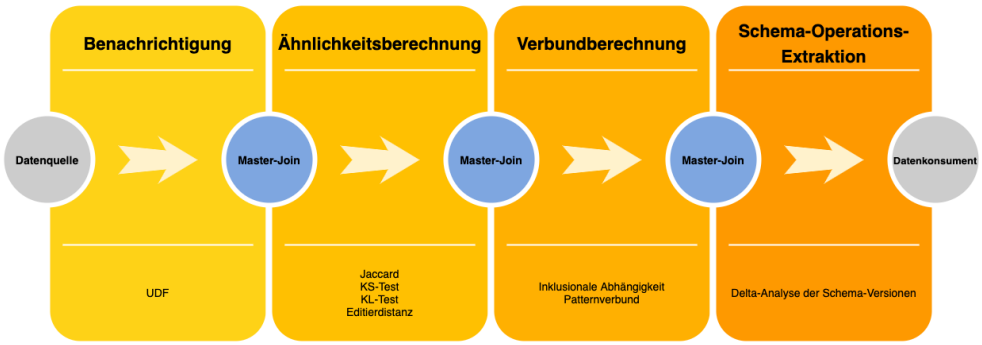


Abbildung 3.2: Aufbau des Master-Joins

Zu den Aufgaben des Master-Joins gehört das Erkennen von Schema-Veränderungen einer Datenquelle. Sobald eine Schema-Veränderung erkannt wurde, werden Ähnlichkeiten und Verbünde berechnet. Schema-Operationen können anschließend extrahiert werden. Jede Erkenntnis wird gespeichert und den Datenkonsumenten zur Verfügung gestellt. Ein Datenkonsument kann für seine Datenmodelle ermitteln, ob Schema-Veränderungen Auswirkungen auf eigene Datenmodelle haben.

ÄHNLICHKEITSMASSE

Die vorgestellten Funktionseigenschaften (vgl. Abschnitt 2.5) können durch die folgenden Ähnlichkeitsmaße erfüllt werden. Diese unterstützen den weiteren Schema-Evolutions-Prozess.

3.2.5 *KS-Test*

Mithilfe des Kolmogorow-Smirnow (KS)-Tests kann eine Wahrscheinlichkeitsverteilung eines metrischen Vektors mit vordefinierten Verteilungsparametern überprüft werden. Dies kann für das Testen einer Stichprobe in einer Fabrik verwendet werden, die überprüfen möchte, ob neue Fabrikate bezüglich der Verteilungsparameter Abweichungen zur Norm aufweisen. Statt die Verteilungsparameter festzuschreiben, können diese auch von einem weiteren metrischen Vektor erhoben werden. Als Verteilung wird die Normalverteilung mit den Verteilungsparametern μ und σ verwendet [29][vgl. 149]. Die Testhypothesen für zwei zu untersuchende metrische Vektoren X und Y sind:

$$H_0 = F_X(x) = F_Y(x)$$
$$H_1 = F_X(x) \neq F_Y(x)$$

Als Signifikanzniveau wird der Parameter α in Abhängigkeit der Sensitivität für die Teststatistik definiert. Wenn die Testhypothesen H_0 nicht signifikant abgelehnt werden kann, bedeutet dies, dass die Wahrscheinlichkeitsverteilungen innerhalb des im Signifikanzniveau liegenden Bereiches gleich sind. Wenn H_0 abgelehnt wird, dann liegen im Rahmen des definierten Signifikanzniveaus, keine Ähnlichkeiten vor [29][vgl. 149].

$$K_n = \sqrt{\frac{n * m}{n + m}} \sup_x |(F_{X,n} - F_{Y,m})(x)|$$

Wenn $K_n > K_{\alpha,n,m}$, wird H_0 signifikant zugunsten von H_1 abgelehnt. In diesem Fall sind die Wahrscheinlichkeitsverteilungen der beiden Vektoren nicht gleich oder

nicht ähnlich genug. Der Ablehnbereich von $K_{\alpha,n,m}$ liegt tabellarisch vor (vgl. Anhang A).

Die Funktionseigenschaften können für den KS-Tests erfüllt werden:

- **Kommutativität**: Die Reihenfolge der Funktionsargumente hat auf den Ähnlichkeitswert keinen Einfluss.
- **Transitivität**: Ist erfüllt, wenn mindestens zwei Attributspaare der Klasse *ähnlich* zugeordnet werden können.
- **Eindeutigkeit**: Der KS-Test ist deterministisch.
- **Skalierbarkeit**: Eine Ähnlichkeitsberechnung kann auf einer Stichprobe durchgeführt werden.
- **Idempotenz**: Die Berechnung eines Ähnlichkeitswertes identischer Attribute führt zu einem Ähnlichkeitswert von 1.
- **Interpretierbarkeit**: In Abhängigkeit der Testhypothesen sind zwei Attribute *ähnlich* oder *disjunkt*.
- **Domänenkompatibilität**: Kann für die Ähnlichkeitsberechnung metrischer Attributsdomänen verwendet werden.

3.2.6 *Beispiel*

Der KS-Test vergleicht zwei Wahrscheinlichkeitsverteilungen miteinander. Die Attribute *score* der beiden *Mission*-Entitäten (Tabelle 2.2 und 2.3) werden wegen ihrer metrischen Domänen für dieses Beispiel gewählt. Die Testhypothesen (vgl. 3.2.5) auf Gleichheit zweier Wahrscheinlichkeitsverteilungen sind zu evaluieren. Anhand der relativen Häufigkeitsverteilung der Attributswerte werden die Entitäten zeilenweise verbunden[4]:

ID	$mission_{v0}$.score	$mission_{v1}$.score	CDF $mission_{v0}$.score[1]	CDF $mission_{v1}$.score[1]	Differenz
1	10	20	$\frac{10}{115} = 0.086$	$\frac{20}{170} = 0.117$	$\lvert 0.086 - 0.117 \rvert = 0.031$
2	5	50	$\frac{5}{115+\frac{10}{115}} = 0.043$	$\frac{50}{170+\frac{20}{170}} = 0.293$	$\lvert 0.043 - 0.293 \rvert = 0.25$
3	100	0	$\frac{100}{115+\frac{5}{115+\frac{10}{115}}} = 0.869$	$\frac{0}{170+\frac{50}{170+\frac{20}{170}}} = 0$	$\lvert 0.869 - 0 \rvert = 0.869$
4	0	100	$\frac{0}{115+\frac{100}{115+\frac{5}{115+\frac{10}{115}}}} = 0$	$\frac{100}{170+\frac{50}{170+\frac{20}{170}}} = 0.587$	$\lvert 0 - 0.587 \rvert = 0.587$
Summe	**115**	**170**	**1**	**1**	**1.737**

[1] Cumulative-Distribution-Function (CDF) = Kumulierte Wahrscheinlichkeitsverteilung

Tabelle 3.5: KS-Test für das Attribut score der Mission-Entität

In Tabelle 3.5 sind die kumulierten Wahrscheinlichkeitsverteilung beider Attribute und deren Differenz zu sehen. Die größte Differenz der Wahrscheinlichkeitsverteilungen liegt bei 0.869. Für das Signifikanzniveau von $\alpha = 0.05$ kann der kritische Wert für das Annehmen/Ablehnen der Testhypothesen aus vorberechneten Listen entnommen werden (vgl. Anhang A). Dieser ist für die 30 betrachteten Attributswerte 0.217. Da die maximale Differenz der kumulierten Wahrscheinlichkeitsverteilungen größer ist als der kritische Wert ($0.869 \geq 0.217$), kann die Testhypothese H_0 zugunsten von H_1 abgelehnt werden (vgl. 3.2.5). Es liegen keine ähnlichen Verteilungen vor.

4 Die relativen Häufigkeitsverteilungen beider Attribute werden absteigend sortiert vermerkt. Gemäß dieser Ordnung werden die Attributswerte beider Entitäten verbunden.

3.2.7 *Jaccard-Index-Numerisch*

Mithilfe des Jaccard-Index-Numerisch-Test $J(X,Y)$ wird die Ähnlichkeit zweier numerischer Vektoren (X,Y) berechnet. Diese müssen ordinal, kategorial oder dichotom skaliert sein [20][vgl. 4].

$$J(X,Y) = \frac{|X \bigcap Y|}{|X \bigcup Y|}$$

Der Jaccard-Index ist das Verhältnis der Schnittmenge $(X \bigcap Y)$ zur Vereinigungsmenge $(X \bigcup Y)$ zweier Vektoren [20][vgl. 4]. Aus der Definition wird ersichtlich, dass keine metrisch skalierten Vektoren betrachtet werden können. Die Vereinigungsmenge zweier metrisch skalierter Vektoren kann durch den potenziell unendlichen Zahlenraum und der Stetigkeit gegen 0 streben. Der Jaccard-Index für metrische Vektoren wird 0. Auch wenn zwei Werte nicht gleich codiert sind, kann eine Ähnlichkeitsbeziehung nicht ausgeschlossen werden. Mithilfe zweier dichotomer Vektoren wird dieses Verhalten demonstriert:

Der erste dichotome Vektor X speichert die Angabe des Geschlechtes mit der Kodierung $\{m,w\}$ ab. Ein zweiter ebenfalls dichotomer Vektor Y benutzt für die semantisch identischen Informationen die Kodierung $\{0,1\}$. Der Jaccard-Index ist 0, da keine Schnittmenge gefunden werden kann.

Die Funktionseigenschaften können für den Jaccard-Index-Numerisch (JIN)-Tests erfüllt werden:

- **Kommutativität**: Die Reihenfolge der Funktionsargumente hat auf den Ähnlichkeitswert keinen Einfluss.
- **Transitivität**: Ist erfüllt, wenn mindestens zwei Attributspaare der Klasse *ähnlich* zugeordnet werden können.
- **Eindeutigkeit**: Der JIN-Test ist deterministisch.
- **Skalierbarkeit**: Eine Ähnlichkeitsberechnung kann auf einer Stichprobe durchgeführt werden.
- **Idempotenz**: Die Berechnung eines Ähnlichkeitswertes identischer Attribute führt zu einem Ähnlichkeitswert von 1.
- **Interpretierbarkeit**: Ein Ähnlichkeitswert zwischen 0 und 1 wird erzeugt.
- **Domänenkompatibilität**: Kann für die Ähnlichkeitsberechnung ordinaler oder nominaler Attributsdomänen verwendert werden.

3.2.8 *Beispiel*

Das Ähnlichkeitsmaß JIN wird für die Untersuchung der Attribute Mission.*pid* und Player.*id* verwendet (Tabelle 2.3 und 2.5). Die Schnittmenge $(pid \bigcap id)$ ist in diesem Beispiel: {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}. Und die Vereinigungsmenge $(pid \bigcup id)$: {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}. Die Mächtigkeit beider Mengen beträgt 15. Es ergibt sich ein Jaccard-Index von $\frac{15}{15} = 1$ und somit gelten die Attribute als ähnlich.

3.2.9 *Jaccard-Index-Alphabetisch*

Für alphabetische Werte kann der Jaccard-Index-Alphabetisch (JIA)-Test verwendet werden. Durch die menschliche Erzeugung von alphabetischen Attributswerten besteht die Gefahr von verrauschten Daten. Diese müssen vor einer Ähnlichkeitsberechnung bereinigt werden. Die Datenbereinigung umfasst drei Schritte [20][vgl. 3]:

1. Stop-Words entfernen

2. Lemmatisierung oder Stemming[5]
3. N-Gramme erzeugen

In alphabetischen Attributen können nicht nur skalare Werte gespeichert werden, sondern auch Sätze, Paragrafen, etc ... Somit besteht ein Eintrag eines Vektors aus mehreren alphabetischen Skalaren, die zu einem Array verbunden werden. Da der Jaccard-Index keine verschachtelten Arrays verarbeiten kann, muss die Datenstruktur aufgefaltet werden. Dies ist möglich, da die Reihenfolge der einzelnen alphabetischen Skalare innerhalb eines Eintrages in einem Vektor durch den Jaccard-Index nicht berücksichtigt wird [7][vgl. 61]. Die daraus erzeugte flache Datenstruktur muss von allen Elementen befreit werden, die keinen Einfluss auf den Ähnlichkeitswert haben. Stop-Words sind alphabetische Werte, die in einer Sprache häufig als Füllwörter verwendet werden [7][vgl. 27]. In der deutschen Sprache sind gängige Stop-Words zum Beispiel: $\{und, ein, wurde, zu, um, da, \dots\}$. Dem anschließend müssen unterschiedliche Grammatiken eines Wortes auf eine gemeinsame Wortstammform zurückgeführt werden. Dies wird durch das Anwenden von Lemmatisierung oder Stemming erreicht. Somit können auch Wörter in unterschiedlichen Grammatiken für das Berechnen der Schnittmenge verwendet werden [7][vgl. 32]. Da verrauschte Daten auch durch die Stop-Words-Entfernung und die Rückführung auf die Wortstammform nicht korrigiert werden können, bedarf es den Einsatz von N-Grammen. Mithilfe von N-Grammen wird ein Wort in mehrere Teilmengen fester Länge (N-Zeichen) unterteilt. Hierbei gibt N die Mächtigkeit der Teilmengen an. Wenn die erforderlichen N-Zeichen erreicht sind, werden diese abgeschlossen und ein neues N-Gramm wird erzeugt. So wird aus dem Wort „Hallo“ mit $N = 3$ die N-Gramm-Menge $\{Hal, all, llo\}$. Zeichendreher und Rechtschreibfehler sind nur in wenigen N-Grammen vertreten [7][vgl. 54]. Die verbleibenden Mengen können für das Berechnen des Jaccard-Index verwendet werden.

Die Funktionseigenschaften können für den JIA-Tests erfüllt werden:

- **Kommutativität**: Die Reihenfolge der Funktionsargumente hat auf den Ähnlichkeitswert keinen Einfluss.
- **Transitivität**: Ist erfüllt, wenn mindestens zwei Attributspaare der Klasse *ähnlich* zugeordnet werden können.
- **Eindeutigkeit**: Der JIA-Test ist deterministisch.
- **Skalierbarkeit**: Eine Ähnlichkeitsberechnung kann auf einer Stichprobe durchgeführt werden.
- **Idempotenz**: Die Berechnung eines Ähnlichkeitswertes identischer Attribute führt zu einem Ähnlichkeitswert von 1.
- **Interpretierbarkeit**: Ein Ähnlichkeitswert zwischen 0 und 1 wird erzeugt.
- **Domänenkompatibilität**: Kann für die Ähnlichkeitsberechnung alphabetischer Attributsdomänen verwendet werden.

3.2.10 *Beispiel*

Die JIA-Ähnlichkeit wird anhand des Attributs *name* der Player Entitäten in der unverfremdeten (Tabelle 2.4) und verfremdeten (Tabelle 2.5) Variante gezeigt. Die Attribute werden mithilfe der beschriebenen Bereinigungsstrategien auf eine gemeinsame Basis gebracht.

- **Stop-Words entfernen**: Im Attribut *name* können keine Stop-Words erkannt werden.

5 Die Lemmatisierung berechnet die Stammform ein Wort mithilfe vordefinierter Wörterbücher. Stemming hingegen ist ein Regelbasierter-Ansatz.

- **Lemmatisierung oder Stemming**: Eine Rückführung auf eine Wortstammform entfällt, da dies bei dem Attribut *name* nicht sinnvoll ist.
- **N-Gramme**: Es werden 3-Gramme verwendet.

Es ergeben sich folgende N-Gramme:

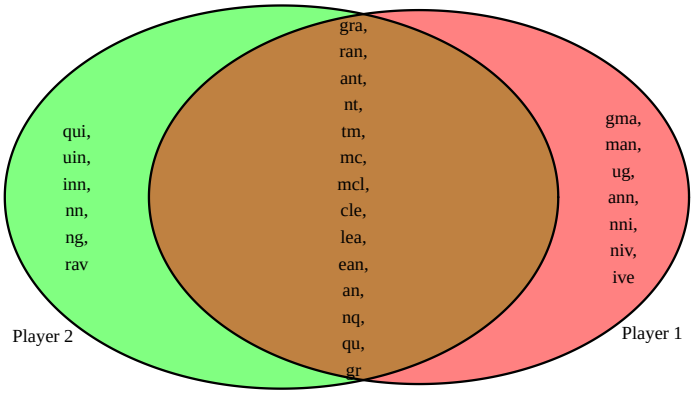


Abbildung 3.3: Stichproben N-Gramm

Die Mächtigkeit der Schnittmenge beträgt 14 und für die Vereinigungsmenge 27. Der Jaccard-Index ist $\frac{14}{27} = 0.51$ und je nachdem welcher Schwellwert θ für die Ähnlichkeitsklassifikation gewählt wurde, sind die Attribute als ähnlich/disjunkt zueinander zu sehen.

3.2.11 *Editierdistanz*

Die Editierdistanz (alt. Levenshtein-Distanz) zweier alphabetischer Wörter ist definiert als die Anzahl an Zeichenänderungsoperationen, die mindestens notwendig sind, um ein Wort in ein anderes zu überführen. Eine Änderungsoperation kann vom Typ *Einfügen*, *Löschen* oder *Ersetzen* sein. Die Berechnung muss über den gesamten Korpus des alphabetischen Attributs durchgeführt werden. Alle möglichen Tupel aus X und Y werden mit der Editierdistanz verglichen. Durch den Schwellwert θ wird die Anzahl der tolerierbaren Änderungsoperationen begrenzt [5, vgl. 7].

$$termsim(X,Y) = \sum_{x \in X} \sum_{y \in Y} edt(x,y) \leq \theta$$

Die Editierdistanz wird über die Funktion edt zwischen den alphabetischen Wörtern X, Y berechnet. Eine Gesamtähnlichkeit ergibt sich durch eine Summation der Einzeleditierabstände aller möglichen Tupel der Attribute X und Y. Eine Normierung von *termsim* auf das Intervall $[0,1]$ ist für eine bessere Interpretierbarkeit notwendig [5, vgl. 7].

$$termsim(X,Y) = \frac{\sum_{x \in X} \sum_{y \in Y} edt(x,y) * max\{|x|,|y|\}}{max\{|X|,|Y|\}} \leq \theta$$

Für zwei skalare Werte x und y gibt $max\{|x|,|y|\}$ die maximale Zeichenlänge zurück. Die Änderungsoperation können unterschiedlich gewichtet werden. Diese Gewichtung ist abhängig von dem zu betrachtenden Kontext. In dieser Arbeit werden alle Änderungsoperationen mit dem Standardgewicht 1 bewertet, da das Ähnlichkeitsmaß nicht auf eine bestimmte Attributskonstellation trainiert werden soll.

Die Funktionseigenschaften können für die Editierdistanz erfüllt werden:

- **Kommutativität**: Die Reihenfolge der Funktionsargumente hat auf den Ähnlichkeitswert keinen Einfluss.
- **Transitivität**: Ist erfüllt, wenn mindestens zwei Attributspaare der Klasse *ähnlich* zugeordnet werden können.
- **Eindeutigkeit**: Die Editierdistanz ist deterministisch.
- **Skalierbarkeit**: Eine Ähnlichkeitsberechnung kann auf einer Stichprobe durchgeführt werden.
- **Idempotenz**: Die Berechnung eines Ähnlichkeitswertes identischer Attribute führt zu einem Ähnlichkeitswert von 1.
- **Interpretierbarkeit**: Ein Ähnlichkeitswert zwischen 0 und 1 wird erzeugt.
- **Domänenkompatibilität**: Kann für die Ähnlichkeitsberechnung alphabetischer Attributsdomänen verwendet werden.

3.2.12 *Beispiel*

Für die Demonstration der Editierdistanz werden die alphabetische Attribute *mission.title* der verfremdeten Entität (Tabelle 2.3, Vektor X) und der unverfremdeten *Mission*-Entität (Tabelle 2.2, Vektor Y) betrachtet. Es wird eine maximale Editierdistanz von zwei gewählt. Alle Änderungsoperationen werden mit 1 gewichtet. Es können folgende Attributswerttupel gefunden werden, die innerhalb der definierten Editierdistanz ähnlich sind:

X	**Y**	**Editierdistanz**
Ski	Ski	0
Ski	Skz	1
Ski	bki	1
Tennis	Tennis	0
Tennis	Teinns	2
Tennis	Tvennis	1
Tennis	Tnnies	2
Puzzle	Puzzle	0
Puzzle	Pzzle	1
Fußball	Fußball	0
Fußball	Fußqball	1
Boxen	oxen	1

Tabelle 3.6: Editierdistanz des Attributs title der Mission-Entität

Mit der Formel aus 3.2.11 wird ein Ähnlichkeitswert von 0.73 berechnet.

3.2.13 *KL-Test*

Da die Restriktionen der bisherigen Ähnlichkeitsmaße hoch sind, muss ein weiteres Ähnlichkeitsmaß eingeführt werden. Die bisherigen Restriktionen sind:

Verfahren	Beschreibung
KS-Test	Es können nur metrische Vektoren betrachtet werden.
JIN und JIA	Wenn unterschiedliche Kodierungen zwischen Vektoren vorliegen, ist der Ähnlichkeitswert 0.
Editierdistanz	Kann nur für gleich kodierte alphabetische Vektoren verwendet werden.

Tabelle 3.7: Restriktionen der Ähnlichkeitsmaße

Durch den Kullback-Leibler-Divergenz (KL)-Test werden alleinig diejenigen Werte für eine Ähnlichkeitsberechnung verwendet, die häufig in einem Datensatz vorhanden sind. Es werden die relativen Häufigkeitsverteilungen der Attributswerte in einem Vektor gespeichert. Anschließend werden die Top-K-Häufigkeitsverteilungen[6] für die Berechnung der statistischen Entropie verwendet [12]:

$$KL(P,Q) = \sum_{x \in X \in topK_X} P(x) * \log(\frac{P(x)}{Q(x)})$$

Mit $P(x)$ wird die relative Häufigkeitsverteilung des Attributswertes x in dem Vektoren P angegeben. Mit dem KL-Test wird die Divergenz zweier Wahrscheinlichkeitsverteilungen untersucht. Die zu untersuchenden Vektoren bilden die Verteilungen P und Q. Gleichheit und Ähnlichkeit kann nur erfüllt sein, wenn $KL(P,Q) \leq 0$ [12].

Die Funktionseigenschaften können für den KL-Test erfüllt werden:

- **Kommutativität**: Die Reihenfolge der Funktionsargumente hat auf den Ähnlichkeitswert keinen Einfluss.
- **Transitivität**: Ist erfüllt, wenn mindestens zwei Attributspaare der Klasse *ähnlich* zugeordnet werden können.
- **Eindeutigkeit**: Der KL-Test ist deterministisch.
- **Skalierbarkeit**: Eine Ähnlichkeitsberechnung kann auf einer Stichprobe durchgeführt werden.
- **Idempotenz**: Die Berechnung eines Ähnlichkeitswertes identischer Attribute führt zu einem Ähnlichkeitswert von 1.
- **Interpretierbarkeit**: Ein Ähnlichkeitswert zwischen 0 und 1 wird erzeugt.
- **Domänenkompatibilität**: Kann für die Ähnlichkeitsberechnung kategorialer Attributsdomänen verwendet werden.

3.2.14 *Beispiel*

Für den Kullback-Leibler-Divergenz (KL)-Test werden zwei nominal oder metrisch skalierte Attribute benötigt. Betrachtet werde die Attribute *score* der beiden *Mission*-Entitäten (Tabelle 2.2 und 2.3). Für jedes Attribut werden die relativen Häufigkeiten der Attributswerte berechnet. Anschließend werden die Attributswerte beider Entitäten gemäß den relativen Häufigkeiten absteigend sortiert und miteinander zeilenweise verbunden. Es gilt die Annahme, dass eine Verteilung eines Attributswertes in der einen Entität bei einer Ähnlichkeitsbeziehung auch eine äquivalente Verteilung in der anderen Entität aufweisen muss.

6 Die Anzahl der zu betrachteten Attributswerte wird mit dem Parameter k gesteuert.

Attributswert	Relative Häufigkeit	Entität
10	$\frac{7}{15} = 0.46$	Mission 1
5	$\frac{4}{15} = 0.26$	Mission 1
100	$\frac{3}{15} = 0.2$	Mission 1
0	$\frac{1}{15} = 0.06$	Mission 1
20	$\frac{5}{15} = 0.33$	Mission 2
50	$\frac{4}{15} = 0.26$	Mission 2
0	$\frac{3}{15} = 0.2$	Mission 2
100	$\frac{3}{15} = 0.2$	Mission 2

Tabelle 3.8: Attributswerte des Attributs score der Mission-Entität für den KL-Test

Mithilfe der Formel aus Abschnitt 3.2.13 kann ein Ähnlichkeitswert ermittelt werden:

$$KL(Mission_{v0}.score, Mission_{v1}.score) =$$
$$0.46 * \log(\frac{0.46}{0.33}) + 0.26 * \log(\frac{0.26}{0.26}) + 0.2 * \log(\frac{0.2}{0.2}) + 0.06 * \log(\frac{0.06}{0.2}) +$$
$$0.33 * \log(\frac{0.33}{0.46}) + 0.26 * \log(\frac{0.26}{0.23}) + 0.2 * \log(\frac{0.2}{0.2}) + 0.2 * \log(\frac{0.2}{0.06})$$
$$= 0.105$$

Auch wieder in Abhängigkeit der Definition des Sensitivitätsparameters für die Ähnlichkeitsklassifikation kann das Ergebnis als signifikant eingestuft werden.

3.3 ÄHNLICHKEITSBERECHNUNG ANGEWANDT

Für einen datengetriebenen Schema-Evolutions-Prozess müssen diejenigen Attribute identifiziert werden, die durch eine Schema-Veränderung verändert worden sind. Wenn eine Multitype-Schema-Operation Auslöser für eine Schema-Veränderung ist, wird eine neue Ähnlichkeitsbeziehung in den Daten erzeugt. Die Ähnlichkeitsbeziehungen können mithilfe von Ähnlichkeitsmaßen isoliert werden. Für unterschiedliche Attributsdomänen werden verschiedene Ähnlichkeitsmaße betrachtet.

3.3.1 *Domänenkompatibilität*

Ein Attribut ist immer eindeutig einer Domänenklasse zugeordnet. Über diese Zuordnung kann anhand einer Domänenkompatibilitätsmatrix ein passendes Ähnlichkeitsmaß für zwei Attribute gefunden werden. Durch unterschiedliche Merkmalsausprägungen ist eine allumfassende Definition eines solchen Maßes nicht sinnvoll. Im Schema-Evolutions-Prozess werden nur Ähnlichkeiten von kompatiblen Attributspaaren berechnet. Die Autoren [8] schlagen die folgende Domänenkompatibilitätsmatrix vor:

	Metrisch	**Ordinal**	**Kategorial**	**Dichotom**
Metrisch	KS-Test[1]	ρ [4]	Regression[5]	Logit[7]
Ordinal	ρ[4]	Jaccard [6]	MANOVA[9]	Rangsummentest[8]
Kategorial	Regression[5]	MANOVA[9]	χ^3	χ^3
Dichotom	Logit[7]	Rangsummentest[8]	ϕ^2	χ^3

[1] KS-Test siehe Abschnitt 3.2.5
[2] Phi-Koeffizient
[3] Chi-Quadrat-Test (Unabhängigkeitstest)
[4] Bravais-Pearson-Korrelation
[5] Lineare Regression mit anschließender Analyse der Reststreuung in der Fehlerzerlegung.
[6] Relativer Anteil der Schnittmenge zur Vereinigungsmenge zweier Vektoren.
[7] Logistische Regression
[8] Nicht parametrische Teststatistik, die den Fokus auf das Bilden von Rängen und deren Relationen legt.
[9] Multivariate Varianzanalyse

Tabelle 3.9: Domänenkompatibilitätsmatrix mit Ähnlichkeitsmaßen aus [8][vgl. 7]

Die von [8] vorgestellten Verfahren können nicht weiter betrachtet werden, da die Autoren nicht berücksichtigt haben, dass jene Verfahren nur verwendet werden können, wenn zusammenhängende[7] Datensätze vorliegen [8]. Im Folgenden wird die Bravais-Pearson-Korrelation betrachtet. Hierbei kann gezeigt werden, dass ein nicht zusammenhängender Datensatz nicht auf Korrelationen untersucht werden kann. Die Bravais-Pearson-Korrelation für zwei ordinalskalierte Vektoren X und Y ist nach [8][vgl. 7] definiert als:

$$\rho_{X,Y} = \frac{cov_{X,Y}}{\sqrt{var_X} * \sqrt{var_Y}}$$

Die Varianzen var können pro Vektor unabhängig voneinander berechnet werden:

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

Die Kovarianz beschreibt den Zusammenhang zwischen X und Y:

$$cov_{x,y} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \frac{1}{n}\sum_{j=1}^{n}(x_j))(y_i - \frac{1}{n}\sum_{k=1}^{n}(y_k))$$

Über die Datensätze wird mithilfe der global gültigen Zählvariable i iteriert. Somit wird davon ausgegangen, dass die Vektoren X und Y verbunden sein müssen. Wenn dies nicht der Fall ist, kann keine globale Reihenfolge innerhalb der Vektoren definiert werden. Ein Vektor kann absteigend sortierte und ein weiterer aufsteigend sortierte Werte haben. Die Korrelation ist somit abhängig von der Sortierreihenfolge der Vektoren. Da dies keinesfalls ausschlaggebend für die Berechnung von Ähnlich-

7 Ein Datensatz ist zusammenhängend, wenn Attributswerte aus einer Entität einer weiteren Entität zugeordnet werden können. Eine solche Zuordnung kann durch einen Datenverbund beschrieben werden.
8 Die Autoren wurden um ein Kommentar zu dieser Problematik gebeten. Dieser liegt leider noch nicht vor. Eventuell äußern sich die Autoren zu einem späteren Zeitpunkt in https://www.researchgate.net/publication/220473778_Instance-based_attribute_identification_in_database_integration/comments

keitswerten sein darf, kann die Korrelation nicht weiter betrachtet werden. Gezeigt wird dies beispielhaft an den Vektoren X, Y und Z, wobei $X = Y = Z$ gilt:

X	Y	Z
1	5	1
2	4	2
3	3	3
4	2	4
5	1	5

Tabelle 3.10: Korrelation von unverbundenen Vektoren

Zwischen den Vektoren X und Y soll der Korrelationskoeffizient berechnet werden: $\rho_{X,Y} = \frac{cov_{X,Y}}{\sqrt{var_X} * \sqrt{var_Y}}$. In der Tabelle 3.10 stehen die Werte für X und Y. Es ergibt sich ein Korrelationskoeffizient von: $\frac{-2.5}{\sqrt{2.5} * \sqrt{2.5}} = -1$. Eine lineare Korrelation kann ausgeschlossen werden. Es gilt, wenn eine lineare Korrelation vorliegt, gelten die Attribute als ähnlich [8][vgl. 13]. Wenn nun der Korrelationskoeffizient zwischen X und Z (aufsteigend sortiert) berechnet wird, ergibt sich ein Korrelationskoeffizient von: $\frac{2.5}{\sqrt{2.5} * \sqrt{2.5}} = 1$ und damit eine hohe lineare Korrelation. Auch der Korrelationsplot zeigt dieses Verhalten in Abbildung 3.4.

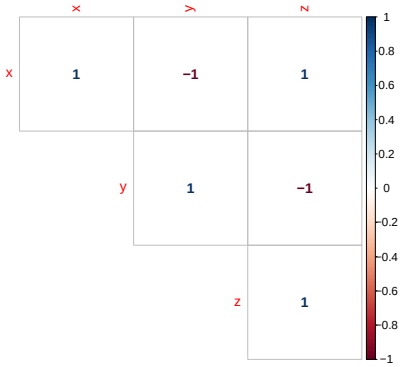


Abbildung 3.4: Korrelationsplot

Neben dem Korrelationskoeffizienten wird die Analyse der Regressionskoeffizienten als Ähnlichkeitsmaß von [8][vgl. 7] vorgeschlagen. Im Folgenden wird eine lineare Regression für zwei numerische Vektoren X und Y beschrieben [15][vgl. 44].

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon_i$$

In der Regression werden Werte für β_0 und β_1 berechnet [15][vgl. 44]. Diese werden folgendermaßen berechnet:

$$\beta_0 = \frac{1}{m} \sum_{k=1}^{m} (y_k)) - \beta_1 \frac{1}{n} \sum_{j=1}^{n} (x_j)$$

$$\beta_1 = \frac{1}{\sum_{i=1}^{n} (X_i - \frac{1}{n} \sum_{j=1}^{n} (x_j))^2} \sum_{i=1}^{n} (X_i - \frac{1}{n} \sum_{j=1}^{n} (x_j))(Y_i - \frac{1}{m} \sum_{k=1}^{m} (y_k))$$

Ein i-ter Wert des Vektors X zeigt auf einen i-ten Wert des Vektors Y. Dies kann mit derselben Argumentation wie in der Problematik mit der Berechnung des Korrela-

tionskoeffizienten nicht sinnvoll berechnet werden. Somit kann die Regression als Ähnlichkeitsmaß nicht betrachtet werden. Alle Verfahren, die einen zusammenhängenden Datensatz benötigen, müssen ausgeschlossen werden.

Es stellt sich die Frage, warum im Schema-Evolutions-Prozess nicht zuerst ein zusammenhängender Datensatz erzeugt wird? Dem anschließend könnte mit der Ähnlichkeitsberechnung begonnen werden. Im Schema-Evolutions-Prozess sind Schema-Veränderungen zwischen Entitäten denkbar, die nicht miteinander in Verbindung stehen. Zum Beispiel kann ein Attribut von einer Entität in eine andere verschoben werden, ohne das eine Beziehung zwischen diesen Entitäten bestehen muss. Wenn ein Entitätsverbund eine notwendige Voraussetzung für die Identifizierung von Multitype-Schema-Operationen wäre, würden diese Operationen nicht erkannt werden. Werden stattdessen zuerst Ähnlichkeiten ermittelt, kann eine Beziehung zwischen den Attributen beider Entitäten gefunden werden. Anschließend kann ein Entitätsverbund berechnet werden. Wenn ein solcher nicht modelliert ist, verbleiben die Erkenntnisse über eine durchgeführte Schema-Operation.

Die Ähnlichkeitsmaße für die Domänenkompatibilitätsmatrix werden gemäß den Einsatzszenarien aus Abschnitt 3.2.4 bestimmt. Die im Abschnitt 2.5.1 beschriebenen Eigenschaften werden durch die Ähnlichkeitsmaße in der Domänenkompatibilitätsmatrix erfüllt:

	Metrisch	**Ordinal**	**Kategorial**	**Dichotom**	**Alphabetisch**
Metrisch	KS-Test	-	-	-	-
Ordinal	-	JIN	JIN	JIN	JIN
Kategorial	-	JIN	KL-Test	JIN	JIN
Dichotom	-	JIN	JIN	JIN	-
Alphabetisch	-	JIA	JIA	-	Editierdistanz/JIA

* Kolmogorow-Smirnow (KS)-Test (vgl. Abschnitt 3.2.5)
* Kullback-Leibler-Divergenz (KL)-Test (vgl. Abschnitt 3.2.13)
* Jaccard-Index-Numerisch (JIN)-Test und Jaccard-Index-Alphabetisch (JIA)-Test (vgl. Abschnitt 3.2.7)
* Editierdistanzt (vgl. Abschnitt 3.2.11)

Tabelle 3.11: Angepasste Domänenkompatibilitätsmatrix mit Ähnlichkeitsmaßen

3.3.2 *Optimierungsstrategien*

Für den Schema-Evolutions-Prozess müssen diejenigen Attribute identifiziert werden, die zueinander ähnlich sind. Für ein Attribut einer Entität müssen alle Attribute für die Ähnlichkeitsberechnung betrachtet werden. Die Menge an zu überprüfenden Attributspaaren definiert den Suchraum. Die Eigenschaften eines Ähnlichkeitsmaßes können für eine Suchraumreduktion verwendet werden:

- **Domänenkompatibilität**
- **Anwendungsbereich**
- **Kommutativität**
- **Transitivität**

Durch das Anwenden der Domänenkompatibilitätsmatrix wird die Anzahl der zu überprüfenden Attributskombinationen reduziert. Hierzu ein kleines Beispiel mit einer vereinfachten Domänenkompatibilitätsmatrix M:

	Ordinal	**Kategorial**	**Dichotom**
Ordinal	x	x	-
Kategorial	x	x	x
Dichotom	-	x	x

Tabelle 3.12: Vereinfachte Domänenkompatibilitätsmatrix M

In einer Datenbank wird eine fiktive Domänenverteilung V erhoben.

	Relative Häufigkeit
Ordinal	0.3
Kategorial	0.4
Dichotom	0.3

Tabelle 3.13: Fiktive Domänenverteilung V

Die Suchraumreduktion kann folgendermaßen berechnet werden:

$1 - (\sum_{dom_a, dom_b \in M} V_{dom_a} * V_{dom_b})$. Alle möglichen Domänenkompatibilitäten werden durch (dom_a, dom_b) betrachtet. Anschließend wird die relative Häufigkeit einer Domäne für dom_a und dom_b aus der Domänenverteilung V ermittelt. Mit der fiktiven Domänenverteilung ist dies:

$$
\begin{aligned}
= 1 - ((V_{Ordinal} * V_{Ordinal}) + (V_{Kategorial} * V_{Kategorial}) + \\
(V_{Dichotom} * V_{Dichotom}) + (V_{Ordinal} * V_{Kategorial}) + \\
(V_{Kategorial} * V_{Dichotom})) \\
= 1 - ((0.3 * 0.3) + (0.4 * 0.4) + \\
(0.3 * 0.3) + (0.3 * 0.4) + \\
(0.4 * 0.3)) \\
= 1 - 0.58 \\
= 0.42
\end{aligned}
$$

42% der Attributspaare müssen nicht mit einem Ähnlichkeitsmaß überprüft werden. Zusätzlich kann durch das Ausnutzen *Kommutativität* der verbleibende Suchraum um die Hälfte reduziert werden, da die Reihenfolge der Funktionsargumente keinen Einfluss auf den Ähnlichkeitswert hat. Mit Kontext transitiver Beziehungen wird der Suchraum reduziert, weil aus bestehenden Ähnlichkeitsbeziehungen auf nicht untersuchte Attributspaare geschlossen werden kann. Durch den *Anwendungsbereich* werden Attribute, die eine gemeinsame Entität teilen, von der Ähnlichkeitsberechnung ausgeschlossen.

3.3.3 *Beispiel*

Der bisherige Schema-Evolutions-Prozess wird mit den Datensätzen aus Abschnitt 3.1 gestartet. Eine Schema-Veränderung ist durch die Initiierungs-Komponente beobachtet worden. Ein Datagramm wird der Vermittler-Komponente mit der Information $data.player_{v0}$ übergeben. Der Master-Join erhält das Datagramm. Nach dem Parsen des Datagramms wird für die $player_{v0}$-Entität eine Basisentität erzeugt. Das Pattern für das Extrahieren der Basisentität ist in diesem Beispiel $[A - Za - z]\{1, \}$. Die Basisentität ist *player* und besteht aus den Entitäten $player_{v0}$ und $player_{v1}$. Da

keine weiteren Entitäten jener Basisentität gefunden werden können, erübrigt sich eine Sortierung nach Schema-Versionen. Mithilfe der Differenzmenge wird der Unterschied zwischen der aktuellen und der vorherigen Schema-Version berechnet. Die Differenzmenge zur Basisentität *player* besteht aus dem Attribut *score*. Dieses ist in einer vorherigen Schema-Version vorhanden ($player_{v0}$) und fehlt in der aktuellen Schema-Version ($player_{v1}$).

Für das Attribut *score* werden Ähnlichkeitswerte berechnet. Für die Optimierung des Suchraums müssen die Domänenklassen für alle Attribute einmalig berechnet werden. Es ergeben sich die Domänenklassen aus Abschnitt 2.7. Da das Attribut *score* in der Schema-Version $player_{v1}$ entfernt wurde, müssen die Attributswerte aus der Schema-Version $player_{v0}$ geladen werden. Anschließend werden alle möglichen Attributspaare generiert und definiere den Suchraum:

- $player_{v0}$.score und $mission_{v1}$.id
- $player_{v0}$.score und $mission_{v1}$.ts
- $player_{v0}$.score und $mission_{v1}$.title
- $player_{v0}$.score und $mission_{v1}$.pid
- $player_{v0}$.score und $mission_{v1}$.score
- $mission_{v1}$.id und $player_{v0}$.score
- $mission_{v1}$.ts und $player_{v0}$.score
- $mission_{v1}$.title und $player_{v0}$.score
- $mission_{v1}$.pid und $player_{v0}$.score
- $mission_{v1}$.score und $player_{v0}$.score

Da die Menge an zu überprüfenden Attributspaaren groß ist (auch schon in diesem Beispiel), muss der Suchraum eingeschränkt werden. Folgende zu überprüfende Attributspaare verbleiben nach dem Anwenden der Optimierungsstrategien:

Berücksichtigung der Kommutativität:

- $player_{v0}$.score und $mission_{v1}$.id
- $player_{v0}$.score und $mission_{v1}$.ts
- $player_{v0}$.score und $mission_{v1}$.title
- $player_{v0}$.score und $mission_{v1}$.pid
- $player_{v0}$.score und $mission_{v1}$.score

Berücksichtigung der Domänenkompatibilität:

- $player_{v0}$.score und $mission_{v1}$.ts
- $player_{v0}$.score und $mission_{v1}$.score

Die Funktionseigenschaften der *Transitivität* und des *Anwendungsbereiches* führen in diesem Beispiel zu keiner Reduzierung des Suchraums. Ein passendes Ähnlichkeitsmaß für die Domänen der zu überprüfenden Attribute ist in der Domänenkompatibilitätsmatrix zufinden (vgl. Tabelle 3.11). In diesem Beispiel muss für zwei ordinalskalierte Attribute ein Ähnlichkeiswert ermittelt werden. Es wird die Jaccard-Ähnlichkeit berechnet:

- $player_{v0}$.score und $mission_{v1}$.ts => 0
- $player_{v0}$.score und $mission_{v1}$.score => 1

Die Ähnlichkeitswerte sind eindeutig, da das Attribut *ts* zum Attribut *score* keine Schnittmenge aufweisen kann. Es ist ersichtlich, dass das Attribut *score* in der vorherigen Schema-Version der *Player*-Entität mit dem Attribut *score* in der aktuellen Schema-Version der *Mission*-Entität ähnlich ist. Das Beispiel wird nach dem Einführen von Verbundfunktionen weitergeführt.

VERBUNDFUNKTIONEN

Die Funktionseigenschaften (vgl. Abschnitt 2.6) können durch die folgenden Verbundfunktionen erfüllt werden und unterstützen den Schema-Evolutions-Prozess:

3.3.4 *Inklusionale Abhängigkeit*

Betrachtet werden zwei Entitäten $e_i, e_j \in E,\ i \neq j$. Inklusionsabhängigkeiten (IND) beschreiben Zusammenhänge zwischen Entitäten e_i, e_j, sodass $e_i[X] \subseteq e_j[Y]$ gilt. Alle Attributswerte aus X der Entität e_i sind auch im Attribut Y der Entität e_j vertreten. Formal gilt: $\forall t_i[X] \in e_i, \exists t_j[Y] \in e_j : t_i[X] = t_j[Y]$ [26][vgl. 2]. Wenn $|X| = |Y| = 1$, dann ist in der zu betrachtenden Attributmenge nur ein Attribut vorhanden wird die IND als unär, andernfalls als k-när bezeichnet. Es wird zwischen dem abhängigen und dem referenzierten Part einer IND unterschieden. $e_i[X]$ ist abhängig von $e_j[Y]$ und wird auch als Left-Hand-Side (LHS) bezeichnet. Die $e_j[Y]$ bildet $e_i[X]$ vollständig ab und wird als referenzierte Seite beziehungsweise als Right-Hand-Side (RHS) bezeichnet [17, vgl. 443]. Für eine Normierung den Wertebereich von $[0, 1]$ wird $\frac{e_i[X] \subset e_j[Y]}{|e_i[X]|}$ berechnet. Zwischen zwei Vektoren X und Y kann nur eine IND bestehen, wenn [18][vgl. 2386]:

- $min(X) \geq min(Y)$
- $min(X) \in Y$
- $max(X) \leq max(Y)$
- $max(X) \in Y$

Diese Ausschlusskriterien können vor der Berechnung der IND überprüft werden. Hierfür können die Statistiken eines DBMS verwendet werden. So kann die Menge an zu überprüfenden IND-Kandidaten reduziert werden. Zusätzlich wird durch die Berücksichtigung der Statistiken kein Datenzugriff benötigt und somit das Laufzeitverhalten verbessert [30][vgl. 417][9]. Durch das Aufdecken von INDs können Primär- und Fremdschlüssel-Beziehungen extrahiert werden. Für den Verbund zweier Entitäten wird eine Verbund-Klausel benötigt. Diese beschreibt, wie Entitäten zeilenweise miteinander verbunden werden können. Zwei Entitäten werden miteinander verbunden, indem definiert wird, wie Attribute der LHS auf Attribute der RHS zugeordnet werden. Es können mehrere solcher Zusammenschlüsse für einen ganzheitlichen Verbund zweier Entitäten verwendet werden.

Zwei Entitäten e_i und e_j können mithilfe von INDs miteinander verbunden werden. Die Attribute $\{a, b\}$ aus der Entität e_i bilden gemeinsam eine Untermenge zum Attribut x der Entität e_j. Es gilt $e_i(a, b) \subseteq e_j(x), i \neq j$. Somit wird für den Verbund eine Verkettung der Attribute a und b benötigt, da diese nur gemeinsam eindeutig auf das Attribut x zeigen können. Dieses Vorgehen wird von [2] vorgeschlagen. Dem widerstrebt jedoch der Gedanke der Datenmodellierung [23][vgl. 37]. Auch wenn durch NoSQL-Datenbanken das Konzept der Normalformen aufgeweicht werden, werden für einen Verbund zweier Entitäten atomare Werte für Primär- und Fremdschlüssel benötigt [11][vgl. 115]. Das Aufdecken von k-nären INDs ist für ein nachträgliches Erforschen von Strukturen zwischen Entitäten wichtig. Für das Finden von Primär- und Fremdschlüssel-Beziehungen reichen unäre INDs aus. Es ist möglich, dass die Attribute a und b für einen Verbund zusammen benötigt werden. Dies gilt genau dann, wenn a und b disjunkte Mengen sind und in der Zielmenge x jeweils einen Gegenpart finden.

9 Jeder vermeidbarer Datenzugriff ist aus Laufzeitsicht zu vermeiden.

Das Konstrukt der Unique-Column-Combination (UCC) wird in [17] vorgeschlagen. Eine Attributmenge $X \subseteq E$ wird um Duplikate und leere Attributswerte bereinigt. Es gilt $\forall t_i, t_j \in e, i \neq j : t_i[X] \neq t_j[X] \ \wedge \forall t \in e, A \in X : t[A] \neq null$ [17, vgl. 444]. Eine UCC kann einen Primärschlüssel darstellen. Dieser ist der Definition nach eindeutig und nicht null [32][vgl. 23]. Da auch weitere Attribute diese Bedingungen erfüllen können, kann eine UCC nur eine notwendige Bedingung für einen Primärschlüssel sein. Eine IND benötigt auf der RHS eine UCC. Somit kann die potenzielle Menge an IND-Kandidaten reduziert werden. Eine UCC muss der Domänenklasse metrisch oder ordinal angehören.

Da mehr als eine IND zwischen zwei Entitäten vorhanden sein kann, muss ein Verfahren zur Vermeidung von False-Positives (FPs) eingeführt werden. Eine IND ist eine notwendige Voraussetzung für eine Primär-Fremdschlüssel-Beziehung. Es können auch Beziehungen zwischen Entitäten auftreten, die den Eigenschaften einer IND genügen, aber keine Primär-Fremdschlüssel-Beziehung erfüllen können. Zum Beispiel kann eine Attributdublette zwischen zwei Entitäten zu einer IND führen. Eine Reduzierung der Menge gefundener INDs muss erfolgen. Mithilfe der Potenzmenge werden alle möglichen Konstellationen der INDs berechnet. An dieser Stelle ist es opportun die Potenzmenge zu bilden, da die Menge an INDs, die die obigen Eigenschaften erfüllen können, gering ist. Die Mächtigkeit der Teilmengen der Potenzmenge kann beschränkt werden. So wird die Menge der zu überprüfenden Tupel reduziert. Anschließend wird überprüft wie viele Zeilen zwischen beiden Entitäten verbunden werden können. Ziel ist die **Maximierung** der Anzahl der Zeilen des Verbundes bei gleichzeitiger **Minimierung** der Anzahl der beteiligten INDs.

Die Funktionseigenschaften können für Inklusionsabhängigkeiten erfüllt werden:

- **Kommutativität**: Die Reihenfolge der Funktionsargumente hat auf den Verbundwert keinen Einfluss.
- **Transitivität**: Ist erfüllt, wenn mindestens zwei Attributspaare der Klasse *verbunden* zugeordnet werden können.
- **Eindeutigkeit**: Die IND ist deterministisch.
- **Skalierbarkeit**: Eine Verbundberechnung kann auf einer Stichprobe durchgeführt werden.
- **Idempotenz**: Die Verbundberechnung identischer Attribute führt zu einem Verbundwert von 1.
- **Interpretierbarkeit**: Ein Verbundwert zwischen 0 und 1 kann erzeugt werden.
- **Domänenkompatibilität**: Kann für eine Verbundberechnung metrischer und ordinalskalierter Attributsdomänen verwendet werden.

3.3.5 *Beispiel*

Bevor ein potenziell teurer Datenzugriff erfolgen muss, kann durch das Betrachten von Statistiken entschieden werden, ob eine IND-Beziehung zwischen zwei Attributen ausgeschlossen werden kann. Zwischen dem Attribut *id* der *Player*-Entität und dem Attribut *pid* der *Mission-Entität* soll eine IND-Beziehung überprüft werden (Tabelle 2.2 und 2.4). Nachdem die Attribute in die Domänenklassen UCC für das Attribut *id* und ordinal für das Attribut *pid* zugewiesen worden sind, werden die Statistiken der Datenbank aktualisiert[10]. Im Folgenden werden Statistiken für die Entscheidung hinzugezogen, ob eine Untersuchung auf INDs sinnvoll ist:

- $min(Mission.pid) \geq min(Player.id) = 1 \geq 1 =$ 👍

10 Es ist zu empfehlen, die Statistiken auszulesen, die durch ein DBMS automatisch erzeugt werden. Diese Informationen werden von den Abfrageoptimierern des DBMS verwendet. Eine mehrfache Belastung durch die Berechnung von Statistiken wird vermieden. Zusätzlich wird die Pflege der Statistiken durch das DBMS verwaltet [30][vgl. 433].

- $min(Mission.pid) \in Player.id = 1 \in Player.id =$ 👍
- $max(Mission.pid) \leq max(Player.id) = 100 \leq 100 =$ 👍
- $max(Mission.pid) \in Player.id = 100 \in Player.id =$ 👍

Ein Datenzugriff kann gerechtfertigt werden. Hierfür wird ein Verbund zwischen den Attributen $Mission.pid \bowtie Player.id$ berechnet und die Anzahl der Einträge gezählt. Diese wird der Anzahl aller möglichen Einträge der RHS ($Mission.id$) gegenübergestellt. Es ergeben sich folgende Erkenntnisse:

Ein Verbund von $Mission.pid \bowtie Player.id$ resultiert in 100 Einträge. Insgesamt hat die RHS ebenfalls 100 Einträge zu verzeichnen. In Abhängigkeit des definierten Schwellwertparameters θ kann ein Verbund zwischen dem Attribut *pid* und *id* nachgewiesen werden:

$$\frac{|Mission.pid \bowtie Player.id|}{|Player.id|} \geq \theta = \frac{100}{100} \geq \theta$$

3.3.6 *Funktionale Abhängigkeit*

Eine Funktionale Abhängigkeit (FD) beschreibt einen Zusammenhang zwischen Attributen. Ein Attribut A ist funktional von Attribut B abhängig, wenn für alle Werte $b \in B$ auf Werte in $a \in A$ geschlossen werden kann. Eine FD ist dann erfüllt, wenn für jedes Tupelpaar $t_i, t_j \in e(U)$ die Bedingungen $t_i[X] = t_j[X]$ und $t_i[A] = t_j[A]$ erfüllt sind. Eine FD $X \rightarrow A$ mit $X \subseteq E$ und $A \in E$ ist auf eine Instanz e einer Entität E gültig, wenn für alle Tupelpaare $\forall t_i, t_j,\ i \neq j | t_i[A] = t_j[A] \wedge t_i[B] = t_j[B]$ gilt [38][vgl. 6]. Es lassen sich FDs nicht für das Verbinden von Entitäten verwenden, da aus der Definition einer FD hervorgeht, dass von einem Wert in X auf immer gleiche Werte in Y geschlossen werden muss [38][vgl. 6]. Die Kenntnis von Werten aus X reichen aus, um Werte in Y zu erhalten. Dieser Schluss kann nur gezogen werden, wenn bereits verbundene Entitäten vorhanden sind. Bei unverbundenen Entitäten, deren Attributswerte in einer beliebigen Reihenfolge gespeichert sein können, ist jegliche Projektion eines X Wertes auf einen Y Wert zufällig. FDs eignen sich zum Beispiel für eine Datenvalidierung. Wenn ein Attribut A den Wert 5 erhält und das Attribut B den Wert 10, kann eine entsprechende Datenvalidierungsregel erzeugt werden. Diese Idee der funktionalen Datenvalidierung wird in [2] vorgestellt.

3.3.7 *Pattern-Verbund*

Da das Konzept der FD nicht auf unverbundene Entitäten angewendet werden kann, jedoch das Konzept der Abbildung zwischen zwei Entitäten in unterschiedlich skalierten oder kodierten Datensätzen wichtig ist, wird das Konzept des Pattern-Verbundes eingeführt. Angenommen zwei Entitäten besitzen jeweils ein Attribut, die die semantisch gleichen Informationen abbilden, jedoch unterschiedlich skaliert oder kodiert sind. Ein Pattern ist eine eindeutige Abbildung eines Attributswertes auf ein anderes. Zwischen zwei Entitäten muss eine Liste an Patterns aus den Datensätzen ermittelt werden. Da ausgeschlossen werden muss, dass zwei Attribute zueinander disjunkt sind oder eine UCC bilden, muss die Anzahl an Gruppierungen[11] beider Attribute erhoben werden. Es wird angenommen, dass wenn eine Gruppe in einem Attribut deutlich überrepräsentiert ist, dies auch in der zu verbindenden Entität der Fall sein muss. Da die Gruppen nach der Anzahl der Elemente absteigend sortiert werden, werden jene Gruppen miteinander verglichen, die nach der Reihenfolge der Sortierung auf der gleichen Ebene stehen. Damit ein Pattern

11 Eine Gruppe besteht aus einer Menge an gleichen Attributswerten und der Mächtigkeit dieser Menge.

ermittelt werden kann, muss die Ähnlichkeit zwischen beiden Attributswerten in den zu vergleichenden Gruppen ermittelt werden. Da der Pattern-Verbund nur für alphabetische Attribute geeignet ist, wird die Editierdistanz als Ähnlichkeitsmaß verwendet. Wenn die Editierdistanz den Schwellwertparameter θ nicht überschreitet, werden die Attributswerte einer Pattern-Liste hinzugefügt.

Die Funktionseigenschaften können für einen Pattern-Verbund erfüllt werden:

- **Kommutativität**: Die Reihenfolge der Funktionsargumente hat auf den Verbundwert keinen Einfluss.
- **Transitivität**: Ist erfüllt, wenn mindestens zwei Attributspaare der Klasse *verbunden* zugeordnet werden können.
- **Eindeutigkeit**: Der Pattern-Verbund ist deterministisch.
- **Skalierbarkeit**: Eine Verbundberechnung kann auf einer Stichprobe durchgeführt werden.
- **Idempotenz**: Die Verbundberechnung identischer Attribute führt zu einem Verbundwert von 1.
- **Interpretierbarkeit**: Ein Verbundwert zwischen 0 und 1 kann erzeugt werden.
- **Domänenkompatibilität**: Kann für eine Verbundberechnung alphabetischer Attributsdomänen verwendet werden.

3.3.8 *Beispiel*

Da für den Pattern-Verbund die Editierdistanz verwendet wird, werden weitere Entitäten für dieses Beispiel eingeführt. Es soll untersucht werden, ob das Attribut *stadt* zweier unterschiedlicher Entitäten semantisch ähnlich sind und anschließend potenziell als Verbund verwendet werden können. In Tabelle 3.14 sind die Entitäten gezeigt. Die Einträge sind gemäß deren relativen Häufigkeiten sortiert.

$Entity_{v1}.stadt$		$Entity_{v2}.stadt$
Frankfurt am Main		Frankfurt a.M.
Frankfurt (Oder)		Frankfurt (O)
Köln		Koeln

Tabelle 3.14: Motivation für einen Pattern-Verbund

Die Entitäten können mithilfe eines Patterns verbunden werden. Dies kommt der Idee der FDs nach, da ein Pattern eine eindeutige Abbildung exakt eines Attributswertes einer Entität auf exakt einen Attributswert einer anderen Entität ist. Eine FD kann nicht beschreiben, wie einzelnen Attributswerte miteinander verbunden werden. Mit einem Pattern-Verbund ist dies möglich. In dem obigen Beispiel ist das Pattern für den Entitätsverbund:

Pattern
$Entity_{v1}.stadt$(Frankfurt am Main) $\Leftrightarrow$ $Entity_{v2}.stadt$(Frankfurt a.M.)
$Entity_{v1}.stadt$(Frankfurt (Oder) $\Leftrightarrow$ $Entity_{v2}.stadt$(Frankfurt (O))
$Entity_{v1}.stadt$(Köln) $\Leftrightarrow$ $Entity_{v2}.stadt$(Koeln)

Tabelle 3.15: Pattern zwischen den Entitäten $Entity_{v1}$ und $Entity_{v2}$

Ein Entitätsverbund wird dann durch das Verbinden der Einzelnen Patterns und der anschließenden Vereinigung aller Ergebnismengen der berechneten Pattern-Verbunde erreicht.

3.4 VERBUNDBERECHNUNG ANGEWANDT

Wenn eine neue Ähnlichkeitsbeziehung innerhalb der Daten beobachtet werden kann, muss für die Vervollständigung einer Multitype-Schema-Operation eine Verbundberechnung durchgeführt werden. Die Entitäten der zueinander ähnlichen Attribute werden für diese Berechnung betrachtet. Zwischen diesen beiden Entitäten werden Attributskombinationen gesucht, mit denen ein Entitätsverbund vollzogen werden kann. Durch die Definition einer weiteren Domänenkompatibilitätsmatrix für die Verbundberechnung kann die Menge an möglichen Verbund-Kandidaten eingeschränkt werden.

3.4.1 *Domänenkompatibilität*

Es wird eine Domänenkompatibilitätsmatrix benötigt, die definiert, welche Domänen für eine Verbundberechnung betrachtet werden können.

	Metrisch	**Ordinal**	**Kategorial**	**Dichotom**	**Alphabetisch**
Metrisch	IND	IND	-	-	-
Ordinal	IND	-	-	-	-
Kategorial	-	-	-	-	-
Dichotom	-	-	-	-	-
Alphabetisch	-	-	-	-	Pattern-Verbund

* Inklusionsabhängigkeiten-Test siehe Abschnitt 3.3.4
* Pattern-Verbund siehe Abschnitt 3.3.7

Tabelle 3.16: Domänenkompatibilitätsmatrix mit Verbundstrategien

Die Domänenkompatibilitätsmatrix der Verbundberechnung ist im Vergleich zur Domänenkompatibilitätsmatrix der Ähnlichkeitsmaße deutlich spärlicher besetzt. Dies ist darauf zurückzuführen, dass für einen Entitätsverbund **ausschließlich** Primär- und Fremdschlüssel-Beziehungen gefunden werden müssen. Eine Eigenschaft eines Primärschlüssels ist die Einmaligkeit der Attributswerte [32][vgl. 23]. Diese Eigenschaft kann für metrische und ordinale Attributswerte erfüllt werden. Durch einen Pattern-Verbund können alphabetische Attribute miteinander verbunden werden. Dies ist notwendig, wenn unterschiedliche Datenkodierungen verwendet werden.

3.4.2 *Optimierungsstrategien*

Berücksichtigt man die vorgestellten Verbundeigenschaften nicht, so muss, um INDs zu bestimmen, für die Attribute beider Entitäten das Kreuzprodukt berechnet werden. Daraus ergeben sich $\binom{n}{2} * 2$-Mengen[12], wobei n die Gesamtanzahl der Attribute ist. Durch das Anwenden der Eigenschaften der Verbundfunktion aus 2.6.2 kann der IND-Suchraum reduziert werden. Im Einzelnen sind dies:

- Durch das Anwenden der Kommutativität wird der Suchraum um den Faktor 0.5 zu $\binom{n}{2}$ vergleichenden Mengen reduziert.
- Der *Anwendungsbereich* lässt ausschließlich Attributspaare unterschiedlicher Entitäten für eine Verbundberechnung zu.
- Wenn eine Menge von Attributen gefunden wird, die auf eine transitive Beziehung schließen lässt, müssen diese Mengen nicht überprüft werden. Alle

12 Ohne die Berücksichtigung der Kommutativität.

gesammelten Erkenntnisse über Entitätsverbünde werden gespeichert. Diese Informationen sind statisch, da eine Veränderung eines Verbundes durch eine Schema-Operation ein schwerwiegender Eingriff in ein bestehendes Datenmodell darstellt.
- Da die RHS eine UCC bilden muss und die LHS nur die ordinale oder nominale Domänenklasse aufweisen kann, reduziert sich der Suchraum gemäß der Häufigkeitsverteilung jener Domänen: $P(UCC) * P(ordinal) + P(UCC) * P(nominal)$
- Zusätzlich kann mithilfe vorhandener Statistiken überprüft werden, ob die minimalen Anforderungen an eine IND erfüllt sind. Ist dies nicht der Fall, kann dieses Attributspaar ohne das Sichten der Daten von der weiteren Verarbeitung ausgeschlossen werden.

3.4.3 *Beispiel*

Der Beispielhafte Schema-Evolutions-Prozess aus Abschnitt 3.1 wird fortgeführt. Mithilfe eines Ähnlichkeitsmaßes ist eine Ähnlichkeitsbeziehung zwischen den Attributen *score* der Entitäten $player_{v0}$ und $mission_{v1}$ aufgedeckt worden. Anschließend wird nun ein Entitätsverbund zwischen den Entitäten $player_{v_1}$ und $mission_{v1}$ benötigt. Dieser wird auf der jeweils aktuellen Schema-Version berechnet. Hieraus ergeben sich folgende Attributspaare[13]:

- $player_{v_1}$.id und $mission_{v1}$.id
- $player_{v_1}$.id und $mission_{v1}$.ts
- $player_{v_1}$.id und $mission_{v1}$.title
- $player_{v_1}$.id und $mission_{v1}$.pid
- $player_{v_1}$.id und $mission_{v1}$.score
- $player_{v_1}$.ts und $mission_{v1}$.id
- $player_{v_1}$.ts und $mission_{v1}$.ts
- $player_{v_1}$.ts und $mission_{v1}$.title
- $player_{v_1}$.ts und $mission_{v1}$.pid
- $player_{v_1}$.ts und $mission_{v1}$.score
- $player_{v_1}$.name und $mission_{v1}$.id
- $player_{v_1}$.name und $mission_{v1}$.ts
- $player_{v_1}$.name und $mission_{v1}$.title
- $player_{v_1}$.name und $mission_{v1}$.pid
- $player_{v_1}$.name und $mission_{v1}$.score
- $player_{v_1}$.address und $mission_{v1}$.id
- $player_{v_1}$.address und $mission_{v1}$.ts
- $player_{v_1}$.address und $mission_{v1}$.title
- $player_{v_1}$.address und $mission_{v1}$.pid
- $player_{v_1}$.address und $mission_{v1}$.score

Für einen Verbund muss auf der RHS eine UCC vorhanden sein. Eine UCC kann nur für ordinal oder nominal skalierte Attribute erfüllt sein. Die LHS darf entweder ordinal, nominal oder UCC skaliert sein. Diese Berücksichtigung führt zu dem folgenden neuen Suchraum:

- $player_{v_1}$.id und $mission_{v1}$.id
- $player_{v_1}$.id und $mission_{v1}$.pid
- $player_{v_1}$.ts und $mission_{v1}$.id
- $player_{v_1}$.ts und $mission_{v1}$.pid

13 Die Kommutativität ist Ausgründen der Darstellung schon berücksichtigt

Zusätzlich können die Datenbankstatistiken ein Indiz liefern, ob ein Verbund möglich ist:

Entität	Attribut	Minimaler Wert	Maximaler Wert
Player	id	1	100
Player	ts	1.559.230.127	1.620.732.335
Mission	id	1	100
Mission	pid	1	100

Tabelle 3.17: Domänenklassen des Mission-Player-Datenmodells

Das Attribut *ts* kann für einen Entitätsverbund nicht weiter betrachtet werden, da der minimale als auch der maximale Wert außerhalb des Wertebereichs der zu vergleichenden Attribute liegen. Der Suchraum beschränkt sich auf die folgenden Attributspaare:

- $player_{v_1}$.id und $mission_{v1}$.id
- $player_{v_1}$.id und $mission_{v1}$.pid

Für die zwei Attribute ist in der Domänenkompatibilitätsmatrix ein Verbundmaß definiert (vgl. Tabelle 3.16). In diesem Beispiel wird die IND berechnet. Es werden folgende Verbundwerte ermittelt:

- $player_{v_1}$.id und $mission_{v1}$.id => 1
- $player_{v_1}$.id und $mission_{v1}$.pid => 1

Mithilfe der Validierung wird versucht, mögliche FPs aus der Menge der Entitätsverbünde zu entfernen. Das Attributspaar $player_{v_1}$.id und $mission_{v1}$.pid kann, wenn diese verbunden werden, die meisten Zeilen miteinander verbinden.

Im abschließenden Schritt kann die ursprüngliche Multitype-Schema-Operation extrahieren werden:

$$MOVE\ player_{v0}.score\ to\ mission_{v_1}.score\ WHERE\ player_{v_1}.id\ =\ mission_{v_1}.pid$$

Die erwartet Multitype-Schema-Operation ist mit der beobachteten Multitype-Schema-Operation identisch. Der Schema-Evolutions-Prozess ist abgeschlossen und die Erkenntnisse können von Datenkonsumenten verwendet werden, um eigene Datenmodellierungen gemäß der durchgeführten Multitype-Schema-Operation anzupassen.

3.5 ALGORITHMISCHER ABLAUF

Die notwendigen Schritte zur Identifizierung von Multitype-Schema-Operationen sind in dem folgenden Pseudocode beschrieben:

Algorithm 1 Master-Join-Verfahren

Require: Datagram d

```
1:  (database, entity) ← extractDatagramParameters(d)
2:  entity ← lookUpEntity(entity, database)
3:  prevEntity ← lookUpPreviousEntity(entity, database)
4:  changedAttribute ← getSchemaDelta(entity, prevEntity)
5:  allPosibleAttributs ← getAllPosibleAttributsInScope()
6:  compatibleAttributs ← getCompatibleAttributs(changedAttribute, allPosibleAttributs)

7:  optimizedAttributeTupelList ← optimize(compatibleAttributs)
8:  for all attribute in optimizedAttributeTupelList do
9:    similarities ← sim(attribute, changedAttribute)
10:   for all similarAttribute in similarities do
11:     lhsEntity ← similarAttribute.lhs.entity
12:     rhsEntity ← similarAttribute.rhs.entity
13:     merges ← merge(lhsEntity, rhsEntity)
14:     for all mergeAttribute in merges do
15:       lhsEntity ← mergeAttribute.lhs.entity
16:       lhsAttribute ← mergeAttribute.lhs.entity.attribute
17:       rhsEntity ← mergeAttribute.rhs.entity
18:       rhsAttribute ← mergeAttribute.rhs.entity.attribute
19:       if changedAttribute is existing in new Entity-Version then
20:         schemaOperation ← COPY
21:       else
22:         schemaOperation ← MOVE
23:       end if
24:     end for
25:   end for
26: end for
27: return $schemaOperation $changedAttribute TO $similarAttribute WHERE
    $lhsEntity.$lhsAttribute = $rhsEntity.$rhsAttribute
```

Mit der Funktion *extractDatagramParameters(datagram)* wird ein eingehendes Datagram geparst und in die Komponenten *database* und *entity* aufgeteilt. Bisherige Erkenntnisse dieser Entität werden durch ein *lookUpEntity(entity, database)* aus der Master-Join-Datenbank geladen. Vorherige Schema-Versionen werden durch das Bilden von Basisentitäten gefunden. Alle Entitäten derselben Basisentität werden gemäß dem Änderungszeitpunkt sortiert. Die aktuelle und die vorherige Schema-Version werden mit der Funktion *lookUpPreviousEntity(entity,database)* ermittelt.

Zwischen der aktuellen und der vorherigen Schema-Version werden diejenigen Attribute extrahiert, denen eine Schema-Veränderung widerfahren ist (Zeile 4). Hierbei ist zu notieren, ob ein Attribut zwischen den Schema-Versionen entfernt oder hinzugefügt wurde. Ein Benutzer kann für eine Reduzierung des Suchraums einen *Scope* definieren. Ein *Scope* beschreibt die Menge der zu betrachtenden Entitäten. Alle Attribute, die diesem *Scope* genügen, werden durch die Funktion *getAllPosible-AttributsInUserScope()* ermittelt.

Anhand der Domänenkompatibilitätsmatrix kann die Menge möglicher Attributspaare reduziert werden (Zeile 6). Zusätzlich werden Optimierungsstrategien angewendet (Zeile 7 und Kapitel 2.5). Die verbleibenden Attributspaare werden der Ähnlichkeitsberechnung *sim(attribute,changedAttribute)* zugeführt. Für zwei ähnliche Attribute werden die Entitäten LHS und RHS ermittelt. Zwischen diesen Entitäten werden Attribute gesucht, die für einen Zeilenverbund (Zeile 13) verwendet werden können. Abschließend werden die Multitype-Schema-Operationen *Move* und *Copy*

extrahiert. Rückgabe ist eine Liste von Multitype-Schema-Operationen, die durch eine eingebrachte Schema-Veränderung, verursacht wurden.

3.6 DATENSCHUTZ

Es ist zu klären, ob die vorgestellte Datenverarbeitung in Einklang mit der Datenschutz-Grundverordnung (DSGVO) zu bringen ist. Neben den wissenschaftlichen Erkenntnissen ist zu prüfen, ob ein produktiver Einsatz möglich ist. Die DSGVO findet nach [35, § 2 Absatz 1] Anwendung in der Verarbeitung personenbezogener digitaler Daten. Somit bestimmt die DSGVO den gesetzlichen Rahmen für die Datenverarbeitung in dieser Arbeit. Personenbezogene Daten sind nach [35, § 4 Absatz 1] Informationen, die für eine Identifizierung einer natürlichen Person notwendig sind. Auch indirekte Zuordnungen zu einer Person fallen unter personenbezogene Daten. Alle Daten, auch personenbezogene Daten, werden mindestens einmal verarbeitet. Konkrete Ausprägungen sind für die Verarbeitung in dieser Arbeit nicht relevant. Nach [35, § 4 Absatz 2] ist die vorliegende Datenverarbeitung auch mit dem Begriff der Verarbeitung der DSGVO zu decken. Eine Verarbeitung ist:

> „Jeden mit oder ohne Hilfe automatisierter Verfahren ausgeführten Vorgang oder jede solche Vorgangsreihe im Zusammenhang mit personenbezogenen Daten wie das Erheben, das Erfassen, [..], das Auslesen, das Abfragen, [..] oder die Verknüpfung, die Einschränkung, das Löschen oder die Vernichtung“ [35, § 4 Absatz 2].

Ohne eine schriftliche Zustimmung zur Datenverarbeitung von personenbezogene Daten durch die betroffene Person ist die vorliegende Verarbeitung nicht durch das DSGVO gedeckt. Eine solche Zustimmung liegt für den Zweck der Schema-Evolution nicht vor. Eine Ausnahme ist in [35, § 89 Absatz 1] definiert:

> „Die Verarbeitung zu im öffentlichen Interesse liegenden Archivzwecken, zu **wissenschaftlichen** oder historischen **Forschungszwecken** oder zu **statistischen** Zwecken unterliegt geeigneten Garantien für die Rechte und Freiheiten der betroffenen Person gemäß dieser Verordnung. Mit diesen Garantien wird sichergestellt, dass technische und organisatorische Maßnahmen bestehen, mit denen insbesondere die Achtung des **Grundsatzes der Datenminimierung** gewährleistet wird. [..] In allen Fällen, in denen diese Zwecke durch die Weiterverarbeitung, bei der die **Identifizierung von betroffenen Personen** nicht oder nicht mehr möglich ist, erfüllt werden können, werden diese Zwecke auf diese Weise erfüllt.“[35, § 89 Absatz 1]

Das Ergebnis der Datenverarbeitung enthält keinerlei personenbezogene Daten und folgt somit dem Grundsatz der Datenminimierung. Zusätzlich liegt ein wissenschaftlicher Zweck beziehungsweise ein statistischer Zweck für die Datenverarbeitung vor.

Zusammenfassend ist die Datenverarbeitung personenbezogener Daten unter Einhaltung des Grundsatzes der Datenminimierung möglich und es bestehen aus datenschutzrechtlicher Sicht keine Hürden für einen produktiven Einsatz der Verfahren[14].

14 Diese Einschätzung basiert auf der laienhaften Interpretation der DSGVO. Für einen produktiven Einsatz sollte für den rechtskonformen Einsatz der Verfahren ein Fachgutachten eingeholt werden.

4 IMPLEMENTIERUNG

In diesem Kapitel werden die Grundlagen des Softwarestacks und eine prototypische Implementierung der Verfahren dieser Arbeit vorgestellt. Zunächst werden die Grundlagen des Softwarestacks Apache Hadoop in Abschnitt 4.1 vorgestellt. Danach wird im Abschnitt 4.3 der Schema-Evolutions-Prozess prototypisch implementiert. Dieses Kapitel schließt mit einem Beispiel ab.

4.1 APACHE HADOOP

> „Apache Hadoop is the de facto framework for processing and storing large quantities of data, what is often referred to as big data. The Apache Hadoop ecosystem consists of dozens of projects providing functionality ranging from storing, querying, indexing, transferring, streaming, and messaging, to list a few.“ [36, vgl. 3]

Die Hadoop-Plattform ist ein Zusammenschluss mehrerer Komponenten der Apache-Software-Foundation. Jede Komponente erfüllt genau eine Aufgabe und lässt sich mit anderen Komponenten integrieren. Anwendungsfälle für den Einsatz der Hadoop-Plattform liegen primär in der Verarbeitung großer Datenmengen. Der Einsatzzweck einer Hadoop-Plattform ist unter dem Stichwort Big-Data geläufig. Herzstück der Hadoop-Plattform ist das verteilte Dateisystem Hadoop-Distributed-File-System (HDFS). Im HDFS werden die Dateien erst in Blöcke definierter Bytelänge aufgeteilt und anschließend auf mehreren Servern gespeichert. Die Blöcke können auf unterschiedlichen Servern redundant vorliegen. Dies ermöglicht den Betrieb eines fehlertoleranten verteilten Dateisystems [36, vgl. 26]. Eine Apache Hadoop-Plattform kann für den Aufbau einer Data-Lake-Architektur verwendet werden [13][vgl. 8]. Eine Data-Lake-Architektur zeichnet sich durch das Speichern, Verarbeiten und Ausleiten unterschiedlicher Daten aus [13][vgl. 14].

4.2 APACHE HIVE

Neben dem verteilten Dateisystem HDFS wird eine Datenbank benötigt, die eine Schnittstelle zum Anfragen jener gespeicherten Daten bereithält. In der Konstellation eines Hadoop-Stacks kann die verteilte NoSQL-Datenbank Apache Hive verwendet werden. Hive ermöglicht es, mithilfe von SQL auf semistrukturierte Daten in einem verteilten Dateisystem zuzugreifen. Hierfür wird das Konzept Schema-On-Read angewendet. Erst im Lesevorgang wird festgelegt, wie die Daten zu interpretieren sind [23][vgl. 202]. Integritätsbedingungen werden nicht unterstützt[1]. Es können Informationen zu Beziehungen zwischen Entitäten nicht erfasst oder erfragt werden [36, vgl. 56].

4.2.1 *Hive-Architektur*

Durch den Hive-Server wird eine JDBC-Schnittstelle für etwaige Datenkonsumenten bereitgestellt. Anfragen werden in Form von SQL-Abfragen gestellt. Diese werden anschließend durch eine Serverkomponente analysiert und ggf. optimiert. Für die

1 Die betrachteten Softwareversionen sind im Anhang C gelistet.

Optimierung der SQL-Abfragen kann auf eigens erzeugte Statistiken zurückgegriffen werden. Über einen Executor werden in Abhängigkeit eines gewählten Dateisystems verteilte Operationen zur Verarbeitung von SQL-Abfragen gestartet werden. Die Ergebnismenge wird der Serverkomponente zurückgemeldet [36, vgl. 209].

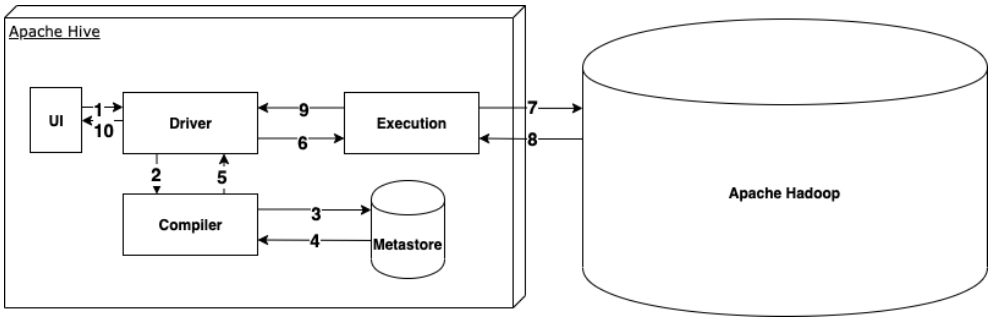


Abbildung 4.1: Apache-Hive Architektur nach [10, vgl. 1] und [36, vgl. 209]

In Abbildung 4.1 sind die Prozessschritte zur Verarbeitung einer DatenbankAnfrage in Hive gezeigt. Über eine Schnittstelle wird im ersten Prozessschritt eine SQL-Abfrage von einen Client an Hive übergeben. Der *Driver* analysiert und koordiniert die weitere Datenverarbeitung. Die SQL-Abfrage wird einem *Compiler* übergeben. Dieser ermittelt im *Metastore* Schema- und Meta-Informationen zur Bearbeitung der SQL-Abfrage. Dem *Driver* werden diese Informationen übermittelt. Dieser übergibt die Datenverarbeitung an einen *Executor*. In Abhängigkeit der gewählten Laufzeitumgebung, zum Beispiel Apache Hadoop, wird die Datenverarbeitung durchgeführt. Das Ergebnis wird dem *Driver* über den *Executor* mitgeteilt. Der *Driver* bereitet das Ergebnis für die Rückmeldung an den Client auf.

Schema-Informationen sind für die Datenverarbeitung durch Hive unerlässlich. Diese Informationen werden im Metastore gespeichert. Wenn eine Schema-Veränderung registriert wird, muss dies dem Metastore mitgeteilt werden. Der Metastore ist eine zentrale Anlaufstelle für die Identifizierung von Schema-Veränderungen.

4.3 PROTOTYPISCHE IMPLEMENTIERUNG

In diesem Abschnitt wird die konzeptionelle Architektur zur Identifizierung von Multitype-Schema-Operationen implementiert.

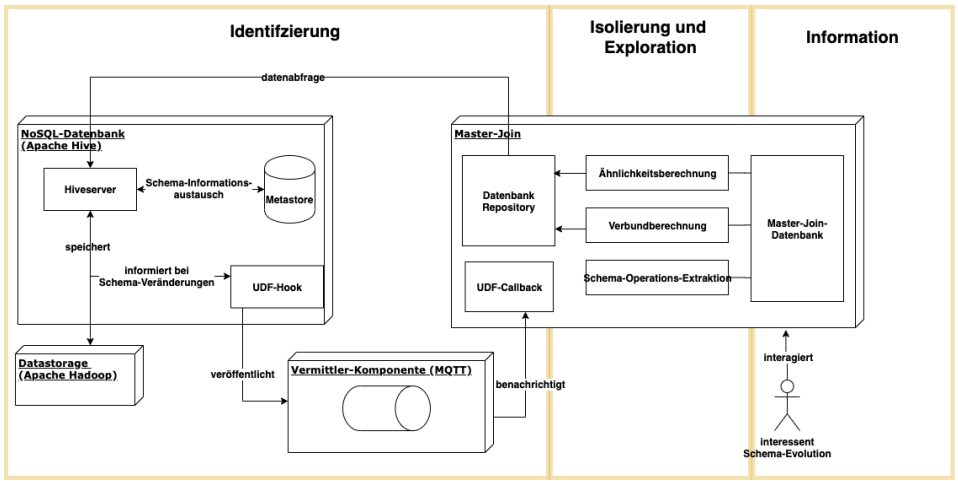


Abbildung 4.2: Implementierte Schema-Evolutions-Architektur

Die Architektur wird im Folgenden komponentenweise eingeführt:

4.3.1 *Identifizierung*

Eine Schema-Veränderung kann nur erkannt werden, wenn ein Schema definiert ist. In der NoSQL-Datenbank Apache Hive kann ein Schema hinterlegt werden. Diese Schema-Informationen werden nicht berücksichtigt, da auch NoSQL-Datenbanken ohne diese Möglichkeit betrachtet werden müssen. Somit muss ein Schema datengetrieben abgeleitet werden. Für eine Entität kann ein Schema durch das Speichern von Attributsnamen und Attributsdomänen erzeugt werden. Mengenwertige Attribute werden durch die Hive-Funktion *explode* in atomare Attribute aufgeteilt [40][vgl. 167]. Anschließend wird die Domänenklassifikation auf einer Stichprobe der Daten durchgeführt. Diese Berechnungen können innerhalb einer Datenbank, zum Beispiel durch eine Datenbankanfrage, implementiert werden. Der Entscheidungsbaum aus Kapitel 2.4 ist zu berücksichtigen. In Hive wird für die Domänenklassifikation eine SQL-Abfrage gemäß den Metadaten aus dem Metastore generiert. Das erzeugte SQL-Statement wird über eine Schnittstelle an Hive übergeben. Die Datenverarbeitung wird verteilt im Cluster ausgeführt. Die erzeugten Schemas sind die Grundlage für die Identifizierung von Schema-Veränderungen. Es existieren zwei Möglichkeiten zur Identifizierung von Schema-Veränderungen [36, vgl. 209]:

- Der bisherige Stand aller Schemas wird in der Anwendung festgehalten. In festen Intervallen werden die aktuellen Schema-Informationen gelesen. Diese Schema-Informationen werden mit früheren Schema-Informationen verglichen. Alle Schema-Veränderungen seit der letzten Ausführung ergeben eine Differenz. Problematisch ist die Definition eines solchen Polling-Intervalls, da ein ständiges vollständiges Lesen aller Schema-Informationen eine zusätzliche Belastung darstellt, die einen operativen Betrieb einer Datenbank beeinträchtigen könnte. Zusätzlich kann die Menge an Schema-Veränderungen mit der Definition des Polling-Invervalls variieren.
- Neben dem aktiven Pollen bietet es sich an, eine Push-Architektur, die lose gekoppelt ist, zu implementieren. Mithilfe einer User-Defined-Function (UDF) kann eine Datenbank erweitert werden und innerhalb dieser Erweiterung kann definiert werden, dass jede Schema-Veränderung zuvor von der definierten UDF validiert werden muss [6][vgl. 163]. Tritt eine Schema-Veränderungen auf, so erhält die Vermittler-Komponente eine Message über die Schema-Veränderung (Push). Im folgenden Abschnitt wir die Vermittler-Komponente eingeführt.

Mit Hive können beide Verfahren implementiert werden. Die Vorteile einer Push-Architektur überzeugen, da nur im Falle eines definierten Events, zum Beispiel einer Schema-Veränderung, eine Message der Vermittler-Komponente übergeben wird. Der Overhead, der entsteht, wenn in festen Intervallen Schema-Veränderungen durch die Polling-Architektur ausgelesen werden, entfällt. Eine Anforderung aus Kapitel 2 ist, dass alle Schema-Veränderungen erkannt und verarbeitet werden müssen. In der Konzeption wird diese Anforderung durch eine Vermittler-Komponente erfüllt. Die Motivation einer solchen Vermittler-Komponente ist, dass mit dem Ausfall von Anwendungen gerechnet werden muss. Damit während eines Ausfalls Schema-Veränderungen nicht unverarbeitet bleiben, werden alle Schema-Veränderungen gespeichert. Mithilfe des Beobachter-Patterns (engl. subscriber pattern) wird der Master-Join benachrichtigt, sobald neue Schema-Veränderungen vorliegen [4][vgl. 219]. Durch die Angabe eines Quality-of-Service (QoS) wird in der Vermittler-Komponente festgelegt, welche Bereitstellungsgarantien erfüllt werden müssen. Es werden folgende Bereitstellungsgarantien unterschieden [16][vgl. 1]:

- **At-most-once**: Ein Datensatz wird **maximal einmal** versendet. Es kann nicht garantiert werden, dass ein Datenkonsument den Datensatz erhält. Ein Netzwerkausfall kann ein Grund für den Verlust und damit ein Ausbleiben der Auslieferung eines Datensatzes sein.
- **At-least-once**: Ein Datensatz wird **mindestens einmal** einem Datenkonsument zur Verfügung gestellt. Eine einmalige Zustellung kann nicht garantiert werden. Ein Datenkonsument muss sich um die Datendeduplikation kümmern.
- **Exactly-once**: Es wird garantiert, dass ein Datensatz **genau einmal** einem Datenkonsumenten zur Verfügung gestellt wird. Im Falle eines Ausfalles ist der Datensatz nicht verloren und wird dann zugestellt, wenn der Normalzustand wiederhergestellt ist.

Es wird die Exactly-Once-Bereitstellungsgarantie benötigt, da eine Datendeduplikation innerhalb der Anwendung nicht implementiert werden kann. Dies ist dem Umstand geschuldet, dass kein Zeitrahmen für ein Zusammenfassen doppelter Einträge definiert werden kann. Da es in einer Data-Lake-Architektur häufig Schema-Veränderungen gibt, wird als Vermittler-Komponente die Middleware MQTT verwendet. Das MQTT-Protokoll ist für einen zuverlässigen Transport vieler Datenpakete optimiert [16][vgl. 1]. Die Abbildung 4.2 zeigt das Zusammenspiel aus Hive, MQTT und Master-Join.

4.3.2 *Isolierung und Exploration*

Ein Datagramm teilt der Vermittler-Komponente eine Schema-Veränderung mit. Nach dessen Erhalt muss festgestellt werden, ob in der Master-Join-Datenbank vorhandene Einträge zu einer solchen Schema-Veränderung gefunden werden können. Für die Differenzmenge zwischen der aktuellen und der vorherigen Schema-Version werden Attributspaare berechnet. Gemäß der definierten Domänenkompatibilitätsmatrix wird entschieden, welche Attributspaare zu untersuchen sind. Die Ähnlichkeitswerte werden nicht in der Anwendung berechnet, da dies den Big-Data-Prinzipien entgegenläuft [23][vgl. 33]. Mithilfe der horizontalen Skalierung werden Datensätze auf unterschiedlichen Servern gespeichert. Um zu vermeiden, dass große Datenmengen über relativ langsame Netzwerke verschoben werden, wird die Logik der Datenverarbeitung zu den Daten gebracht [36][vgl. 52]. Der Master-Join generiert basierend auf den eigenen Schema-Informationen SQL-Abfragen, die verteilt in einem Cluster ausgeführt werden. Das Ergebnis der SQL-Abfrage ist ein Ähnlichkeitswert. Jeder berechnete Ähnlichkeitswert wird in der Master-Join-Datenbank festgehalten. Somit kann eine weitere Abfrage ohne einen Datenzugriff beantwortet werden. Anschließend werden Verbundbeziehungen zwischen den Entitäten berechnet. Für die Verbundberechnung werden die Hive-Statistiken aktualisiert. Der berechnete Verbundwert wird in der Master-Join-Datenbank festgehalten. Die Ähnlichkeits- und Verbundbeziehungen können abschließend für das Bilden von Multitype-Schema-Operationen verwendet werden. Durch die Analyse des Deltas zwischen der aktuellen und der vorherigen Schema-Version kann zwischen den Multitype-Schema-Operationen *Move* und *Copy* unterschieden werden.

4.3.3 *Information*

Ein Datenkonsument kann die erkannten Schema-Veränderungen für eigene Anpassungen verwenden.

4.3.4 *Beispiel*

Im vorherigen Kapitel wurde die Architektur für das Erkennen von Multitype-Schema-Operationen beschrieben und anhand des *Mission-Player*-Datenmodells erläutert. Im folgenden Beispiel werden die Schritte für das Erkennen von Multitype-Schema-Operationen anhand der prototypischen Implementierung nachvollzogen. Es wird die folgende Multitype-Schema-Operation ausgeführt:

$$MOVE\ player_{v0}.score\ to\ mission_{v_1}.score\ WHERE\ player_{v_1}.id\ =\ mission_{v_1}.pid$$

Datendefinition

In den Abbildungen 4.1 und 4.2 werden Hive-Tabellen für die *Player-* und *Mission*-Entitäten definiert. Die Daten sind im HDFS gespeichert und liegen im Datenformat JSON vor. Das Datenformat ist für die Verarbeitung durch Hive unerheblich, da durch die Definition eines Schemas bestimmt wird, wie jene Daten zu lesen sind. Für jede Entität muss eine Tabelle definiert werden.

Listing 4.1: Apache Hive - Externe Tabelle für die Player-Entitäten

```
1  CREATE EXTERNAL TABLE Player_v0 (
2  'ts' int,
3  'id' int,
4  'name'  string,
5  'score'  int,
6  'address' string );
7
8  CREATE EXTERNAL TABLE Player_v1 (
9  'ts' int,
10 'id' int,
11 'name'  string,
12 'address' string )
```

Listing 4.2: Apache Hive - Externe Tabelle für die Mission-Entitäten

```
1  CREATE EXTERNAL TABLE Mission_v0 (
2  'ts' int,
3  'id' int,
4  'title'  string,
5  'pid'  int
6  );
7
8  CREATE EXTERNAL TABLE Mission_v1 (
9  'ts' int,
10 'id' int,
11 'title'  string,
12 'score'  int,
13 'pid'  int
14 );
```

Auch wenn Hive den Begriff *Tabelle* verwendet, so sind Tabellen nach der Definition aus Kapitel 2 Entitäten. Im Kontext von Hive werden die Begriffe *Tabelle* und *Entität* synonym verwendet [34][vgl. 2].

Identifizierung

Hive kann mithilfe von UDFs erweitert werden. Innerhalb dieser Erweiterung kann definiert werden, dass jede Schema-Veränderung zuvor durch eine UDF validiert

werden muss. Die UDF muss die Funktionen des Interfaces 4.3 implementieren. Abschließend wird die UDF Apache Hive mithilfe von 4.4 bekannt gemacht.

Listing 4.3: MetaStoreEventListener

```
interface MetaStoreEventListener {
    void onCreateTable(CreateTableEvent tableEvent) throws MetaException
    void onAlterTable(AlterTableEvent tableEvent) throws MetaException
    void onDropTable(DropTableEvent tableEvent) throws MetaException
    void onDropDatabase(DropDatabaseEvent dbEvent) throws MetaException
    void onCreateDatabase(CreateDatabaseEvent dbEvent) throws MetaException
}
```

Listing 4.4: MetaStoreEventListener

```
add jar /opt/apache-hive-3.1.2-bin/udf/ApacheHiveHook-1.0-SNAPSHOT.jar;
set hive.metastore.event.listeners=edu.hda.se.hivehook.HiveChangeHook;
```

In der UDF wird ein Datagramm für die Schema-Veränderung erzeugt. Dieses ist:

$$data.player_{v1} \text{ und } data.mission_{v1}$$

Das Datagramm wird der Vermittler-Komponente MQTT übergeben.

Isolierung

Die Vermittler-Komponente MQTT übergibt der Anwendung Master-Join das Datagramm in der Exactly-Once-Bereitstellungsgarantie. Der Master-Join ist die Hauptkomponente für das Extrahieren von Multitype-Schema-Operationen. Das Datagramm wird in die Komponenten *Datenbank* und *Entität* aufgeteilt. In der Verarbeitungsphase *Isolierung* werden bereits vorhandene Schemas geladen. Dies ist möglich, da in der Initialisierungsphase des Master-Joins alle Schemas einmalig erzeugt und gespeichert werden. Das Schema wird durch das Generieren von Domänenklassen erzeugt. Für die beiden Entitäten $player_{v1}$ und $mission_{v1}$ werden die Basisentitäten *player* und *mission* ermittelt. In der Master-Join-Datenbank sind Entitäten derselben Basisentität vorhanden. Diese werden geladen und gemäß dem Änderungszeitpunkt sortiert. Für die weitere Verarbeitung sind alleinig die aktuelle und die vorherige Schema-Version einer Entität wichtig. Die Differenzmenge wird zwischen diesen Schema-Versionen gebildet und in der nächsten Verarbeitungsphase verwendet.

Exploration

In der Differenzmenge wird ersichtlich, dass dem Attribut *score* eine Schema-Veränderung widerfahren ist. Im Master-Join wird der Suchraum für die Ähnlichkeitsberechnung aufgebaut. Mithilfe der Optimierungsstrategien kann der Suchraum reduziert werden. Diese Reduzierung kann innerhalb des Master-Joins auf dessen Schema-Informationen durchgeführt werden. Alle Schema-Informationen sind in der Master-Join-Datenbank gespeichert. Der Suchraum (inkl. Optimierung) ist mit dem Suchraum aus Abschnitt 3.3.3 identisch.

Gemäß der Domänenkompatibilitätsmatrix wird die Jaccard-Ähnlichkeit für das Attribut *score* berechnet. Der Master-Join erzeugt für Hive SQL-Abfragen. Diese werden anschließend verteilt im Hadoop-Cluster verarbeitet. Die SQL-Abfrage ist als Template definiert. Zur Laufzeit werden die zu untersuchenden Attribute in das Template eingefügt. In diesem Beispiel wird die folgende SQL-Abfrage erzeugt:

Listing 4.5: SQL-Abfrage der Jaccard-Ähnlichkeit

```
CREATE TEMPORARY TABLE IF NOT EXISTS sampleLHS AS SELECT score  FROM data.
    player_v0 TABLESAMPLE();
CREATE TEMPORARY TABLE IF NOT EXISTS sampleRHS AS SELECT score  FROM data.
    mission_v1 TABLESAMPLE();

with union_ as (
 select  count(*) as unionCount
        from sampleLHS lhs
        full outer join sampleRHS rhs
        ON (lhs.score = rhs.score)
),intersection_ as (
        select  count(distinct lhs.score) as intersectionCount
        from sampleLHS lhs
        inner join sampleRHS rhs
        ON (lhs.score = rhs.score)
    )
select intersectionCount/unionCount as jaccard from union_, intersection_
```

Die SQL-Abfrage berechnete für das Attribut *score* der Entitäten $player_{v0}$ und $mission_{v1}$ einen Ähnlichkeitswert von 1. Diese Erkenntnis wurde der Master-Join-Datenbank hinzugefügt. Für die verbleibenden Attribute des Suchraums konnten keine Ähnlichkeiten ermittelt werden.

Zwischen den Entitäten $player_{v0}$ und $mission_{v1}$ wurde ein Verbund berechnet und ein neuer Suchraum aufgebaut. In diesem Suchraum wurden einzig die Attribute der beiden Entitäten gefunden, für die ein ähnliches Attribut vorhanden gewesen war. Auch dieser Suchraum (inkl. Optimierung) war mit dem Suchraum aus Abschnitt 3.4.3 identisch. Anhand der Domänenkompatibilitätsmatrix wurde bestimmt, dass eine IND berechnet werden musste. Die SQL-Abfrage zur Berechnung von INDs war als Template definiert. Zwischen den beiden Entitäten wurde überprüft, ob die Attribute $player_{v0}.id$ und $mission_{v1}.pid$ eine IND bilden. Das folgende SQL-Template wurde im Hadoop-Cluster ausgeführt:

Listing 4.6: SQL-Abfrage des IND-Verbundes

```
CREATE TEMPORARY TABLE IF NOT EXISTS sampleLHS AS SELECT pid FROM data.
    mission_v1 TABLESAMPLE;
CREATE TEMPORARY TABLE IF NOT EXISTS sampleRHS AS SELECT id FROM data.player_v0
    TABLESAMPLE;

with totalLenght as (
    select count(pid) as tlt
    FROM sampleLHS
    where pid is not null
),
     isIND as (
         select cntInd / tlt  as ind
         FROM (
                  SELECT  count(lhs.pid)  as cntInd
                  FROM sampleLHS lhs
                  JOIN sampleRHS rhs
                  ON (lhs.pid = rhs.id)
              ) ind,
              totalLenght)
select ind
from isIND
```

Es wurde ein Verbundwert von 1 zwischen den Attributen $mission_{v_1}.pid$ und $player_{v_1}.id$ ermittelt. Abschließend konnte die Multitype-Schema-Operation isoliert werden. Diese war:

$$MOVE\ player_{v0}.score\ to\ mission_{v_1}.score\ WHERE\ player_{v_1}.id\ =\ mission_{v_1}.pid$$

Information

Die Multitype-Schema-Operation wurde erkannt. Alle Erkenntnisse wurden in der Master-Join-Datenbank gespeichert. Diese Informationen konnten von Datenkonsumenten verwendet werden, um eigene Datenmodelle gemäß der durchgeführten Schema-Veränderung anzupassen.

5 EVALUATION

In diesem Kapitel wird der vorgestellte Schema-Evolutions-Prozess, bestehend aus den Schritten Ähnlichkeitsberechnung, Verbundberechnung und Extraktion von Multitype-Schema-Operationen, evaluiert. Hierfür werden zwei Datenmodelle verwendet. Ein sehr spezifisches Datenmodell der Commerzbank und das generische TPC-H-Datenmodell. Da das Datenmodell der Commerzbank nur in eben jener Testumgebung evaluiert werden darf, werden zwei Testumgebungen benötigt. Das Datenmodell des TPC-H-Benchmarks wird, soweit nicht anders definiert, lokal in einer eigens erzeugten Testumgebung getestet.

5.1 TESTOBJEKTE

Bevor Testobjekte definiert werden können, müssen Anforderungen an die Datenmodelle gestellt werden:

- Das Datenmodell muss einen Sachverhalt realistisch abbilden.
- Neben der realitätsnahen Modellierung ist das Datenvolumen für die Evaluierung der Skalierung wichtig. Ein Einsatz darf nicht nur auf wenigen ausgewählten Datensätzen möglich sein.
- Insbesondere muss für die Evaluation der Korrektheit der Verfahren bekannt sein, welche Entitäten und Attribute miteinander in Beziehung stehen.

Das Datenmodell des TPC-H-Benchmarks kann diese Anforderungen erfüllen. Der TPC-H-Benchmark entstammt einer Familie von Benchmarks, die unterschiedliche Einsatzszenarien für Datenbanken modellieren und evaluieren [33][vgl. 12]. Die Spezifikation des TPC-H-Benchmark beschreibt dessen Einsatzszenario wie folgt:

> „Other TPC benchmarks model the operational end of the business environment where transactions are executed on a real time basis. The TPC-H benchmark, however, models the analysis end of the business environment where trends are computed and refined data are produced to support the making of sound business decisions. In OLTP benchmarks the raw data flow into the OLTP database from various sources where it is maintained for some period of time.“ [33][vgl. 12]

Diese Eigenschaften decken sich mit den Eigenschaften einer Data-Lake-Architektur [13][vgl. 13]. In Abbildung 5.1 sind die Entitäten und Beziehungen des TPC-H-Datenmodells gezeigt. Da die Beziehungen bekannt sind, kann in der Evaluation die Korrektheit der Ergebnisse verifiziert werden. Zusätzlich wird durch die Spezifikation des Datenmodells in [33] für jedes Attribut beschrieben, wie diese syntaktisch und semantisch zu interpretieren sind. Somit kann zusätzlich evaluiert werden, ob eine Ähnlichkeitsbeziehung zwischen zwei Attributen existiert. Neben der realistischen Datenmodellierung und dem Wissen über Beziehungen der Entitäten kann die Anforderung des Datenvolumens gezielt gesteuert werden. In [33] werden Verfahren für das Generieren von Testdaten beschrieben. Das Datenvolumen ist beliebig und als Parameter zu definieren. Zusätzlich werden alle beschriebenen syntaktischen und semantischen Eigenschaften im Generierungsprozess berücksichtigt.

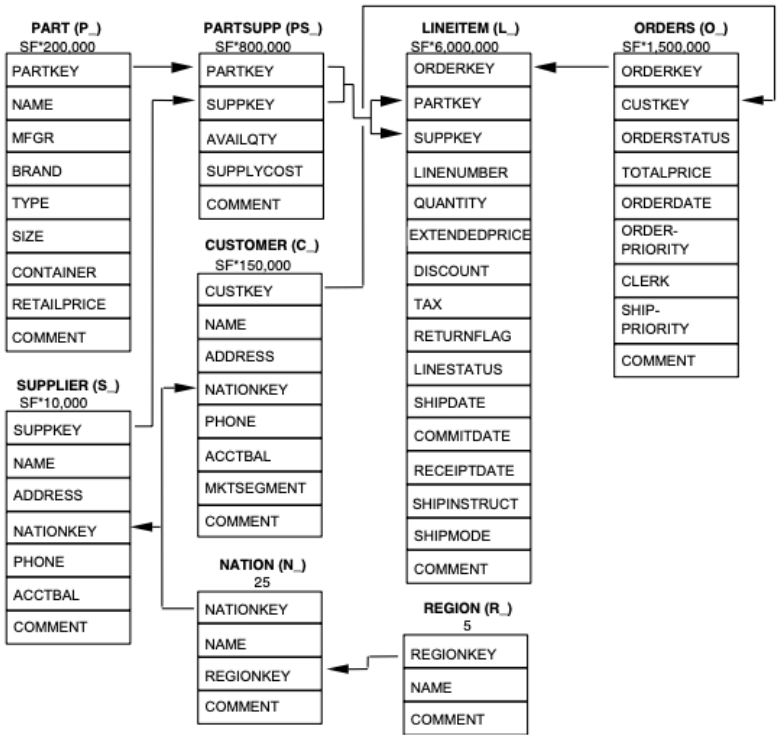


Abbildung 5.1: Datenmodell des TPC-H-Benchmark aus [33][vgl. 13]

Auch wenn das TPC-H-Datenmodell alle gestellten Anforderungen erfüllen kann, wird ein weiteres Datenmodell für eine Evaluation im Kontext der Commerzbank benötigt. Die Anforderung nach Praxisnähe ist für jedes Datenmodell der Bank erfüllt. Auch das Datenvolumen stellt keine Herausforderung dar (i.d.R. ist eine Datenhistorie von bis zu 10 Jahren vorhanden). Die Anforderung an die Korrektheit kann damit befriedigt werden, dass Datenmodelle gewählt werden, deren Spezifikationen ausführlich und vollständig vorliegen. Alle Anforderungen können durch die Datenmodelle *CORE* und *KK-BUCHUNG* (interner Name n3a78c10) erfüllt werden. Da die Semantik des Datenmodells und die konkreten Daten für diese Arbeit nicht relevant sind, werden die Datenmodelle nicht im Detail erläutert. Allgemein handelt es sich bei *CORE* um das Personenstammdatensystem der Commerzbank und bei *KK-BUCHUNG* um Kontotransaktionen und deren Metadaten. Zusätzlich sind beide Datenmodelle eine wichtige Grundlage für einen Großteil der Datenanalysen der Bank und sind gut spezifiziert und dokumentiert.

Aus den Datenmodellen können Testobjekte abgeleitet werden. In Tabelle 5.1 sind alle Testobjekte für diese Evaluation aufgeführt. Zusätzlich wird die Verteilung der Attribute auf die unterschiedlichen Domänenklassen gezeigt.

Testobjekt	Umgebung	Zeilen	Ordinal	Dichotom	Kategorial	UCC	Datum	None	Metrisch	Alphabetisch	Summe
exp_abweichende_gpkdnr	Commerzbank	1.750.604	2	3	1	2	0	0	0	0	8
exp_gp	Commerzbank	23.430.150	2	11	0	3	0	3	0	0	19
exp_gp_organkredit	Commerzbank	58	2	4	0	1	0	0	0	0	7
exp_gp_struktur	Commerzbank	4.014.060	3	3	0	2	0	1	0	0	9
exp_kdnr	Commerzbank	14.723.663	14	7	2	1	0	2	0	5	31
exp_kdnr_adresse	Commerzbank	20.155.772	7	10	2	0	0	0	0	1	20
exp_kdnr_berater	Commerzbank	11.530.591	4	4	0	1	0	0	0	1	10
exp_party	Commerzbank	26.342.902	26	10	1	3	0	17	0	2	59
exp_party_adresse_e	Commerzbank	23.467.250	5	6	1	0	0	2	0	0	14
exp_party_adresse_p	Commerzbank	18.350.627	5	5	1	2	0	2	0	4	19
exp_party_legi_orga	Commerzbank	928.324	5	3	0	1	0	0	0	2	11
exp_party_legi_person	Commerzbank	19.476.392	5	4	0	2	0	1	0	4	16
exp_struktur	Commerzbank	16.863.289	7	6	1	4	0	4	0	2	24
exp_verbund	Commerzbank	248.816	2	6	1	3	1	3	0	1	17
exp_verbund_struktur	Commerzbank	634.197	3	3	0	2	0	0	1	0	9
n3a78c10_saldo	Commerzbank	1.760	24	7	1	0	0	0	2	2	36
n3a73c10_umsatz_kernumsatz	Commerzbank	15	0	0	0	0	0	49	0	0	49
n3a78c10_saldo	Commerzbank	17.002	24	7	1	0	0	0	2	2	36
n3a73c10__cons	Commerzbank	32.636.108	19	11	4	2	0	4	2	7	49
customer	TPC-H	1.500.000	0	0	2	1	0	1	1	3	8
lineitem	TPC-H	29.999.795	3	1	6	0	3	0	2	1	16
nation	TPC-H	25	0	0	0	0	0	4	0	0	4
orders	TPC-H	7.500.000	2	1	2	0	1	0	1	2	9
part	TPC-H	1.000.000	1	0	2	1	0	0	1	4	9
partsupp	TPC-H	4.000.000	3	0	0	0	0	0	1	1	5
revenue	TPC-H	50.000	1	0	0	0	0	0	1	0	2
supplier	TPC-H	50.000	0	0	1	1	0	1	1	3	7
region	TPC-H	5	0	0	0	0	0	5	0	0	5

Tabelle 5.1: Testobjekte TPC-H und Commerzbank

5.2 TESTUMGEBUNG

Es wird eine Apache Hadoop-Plattform in Kombination mit der NoSQL-Datenbank Apache Hive verwendet. Da die verfügbaren Ressourcen der lokalen Testumgebung im Vergleich zu denen der Testumgebung der Commerzbank wesentlich verschieden sind, müssen diese einzeln aufgelistet werden. Eine Herausforderung ist die Ressourcenberechnung in einer mit weiteren Benutzern geteilten Testumgebung. Neben den verfügbaren Ressourcen müssen auch die zugewiesenen/erhaltenen Ressourcen betrachtet werden. Entscheidend für die Beurteilung der Ressourcen sind der Hauptspeicher und die CPU [36][vgl. 17]. In Tabelle 5.2 sind die Ressourcen für die lokale Testumgebung gelistet. Eine Besonderheit kommt der Testumgebung der Commerzbank zu: Es existieren mehrere Server, die für eine Berechnung verwendet werden können. Die Anzahl der Ressourcen ist mit der Anzahl der Server zu multiplizieren. In Tabelle 5.3 sind die kumulierten Ressourcen gelistet.

Ressource	**Verfügung**	**Zugewiesen**
Arbeitsspeicher	32 GB	15 GB
CPU	2,6 GHz 6-Core	2,6 GHz 4-Core

Tabelle 5.2: Ressourcenaufteilung der lokalen Testumgebung

Ressource	**Verfügung**	**Anzahl**	**Gesamtanzahl**
Arbeitsspeicher	755 GB	8	6.040 GB
CPU	2.6 GHz 56-Core	8	2.6 GHz 448-Core

Tabelle 5.3: Ressourcenaufteilung der Commerzbank-Testumgebung

Im Gegensatz zu den Ressourcen der lokalen Testumgebung handelt es sich bei den Ressourcen der Commerzbank um einen theoretischen Wert. Dieser kann wenig über die zu erhaltenden Ressourcen aussagen. Dies liegt an dem Konzept der Ressourcenverteilung durch einen Scheduler[1]. Ein Scheduler teilt die Gesamtanzahl der möglichen Ressourcen (Arbeitsspeicher und CPU) in Queues ein. Eine Queue ist ein prozentual definierbarer Anteil der Gesamtressourcen. Die Ressourcen einer Queue können wiederum von mehreren Benutzern in Anspruch genommen werden. So lässt sich in einer verteilten Umgebung der Ressourcenzugriff mehrerer Benutzer steuern [36][vgl. 18]. Für die Evolution konnte eine Queue mit $20\% - 30\%$ der Gesamtressourcen verwendet werden. Von diesen Ressourcen konnten im Durchschnitt über die 7 Tage der Evaluation 20% verwendet werden. In Abbildung 5.2 ist der Verlauf der erhaltenen Queue-Ressourcen gezeigt. In Tabelle 5.4 sind die erhaltenen Ressourcen aufgelistet.

Ressource	**Gesamtanzahl**	**QK[1]**	**Min. QR[2]**	**Max. QR[2]**	**Min. erhaltene QR[2]**	**Max. erhaltene QR[2]**
Arbeitsspeicher	6.040 GB	$20\% - 30\%$	1.208 GB	1.812 GB	241,6 GB	362,4 GB
CPU	448-Core	$20\% - 30\%$	89-Core	134-Core	18-Core	27-Core

[1] **QK**=Queue-Kapazität
[2] **QR**=Queue-Ressourcen

Tabelle 5.4: Erhaltene Ressourcen der Commerzbank-Testumgebung

1 Es wird der Ressourcenscheduler Apache Hadoop YARN verwendet.

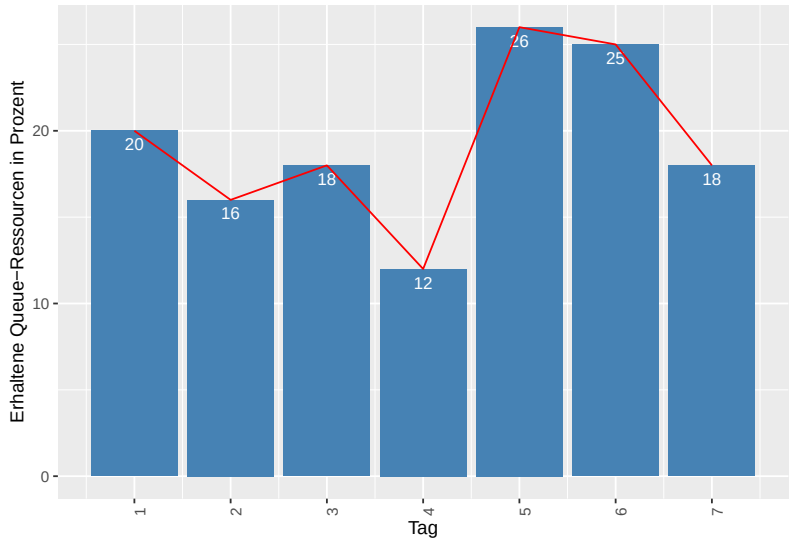


Abbildung 5.2: Erhaltene Queue-Ressourcen

Die erhaltenen Ressourcen liegen deutlich unter den verfügbaren Ressourcen. Gleichzeitig liegen die erhaltenen Ressourcen deutlich über den Ressourcen der lokalen Testumgebung. In Abbildung 5.3 wird dieser Dimensionsunterschied ersichtlich.

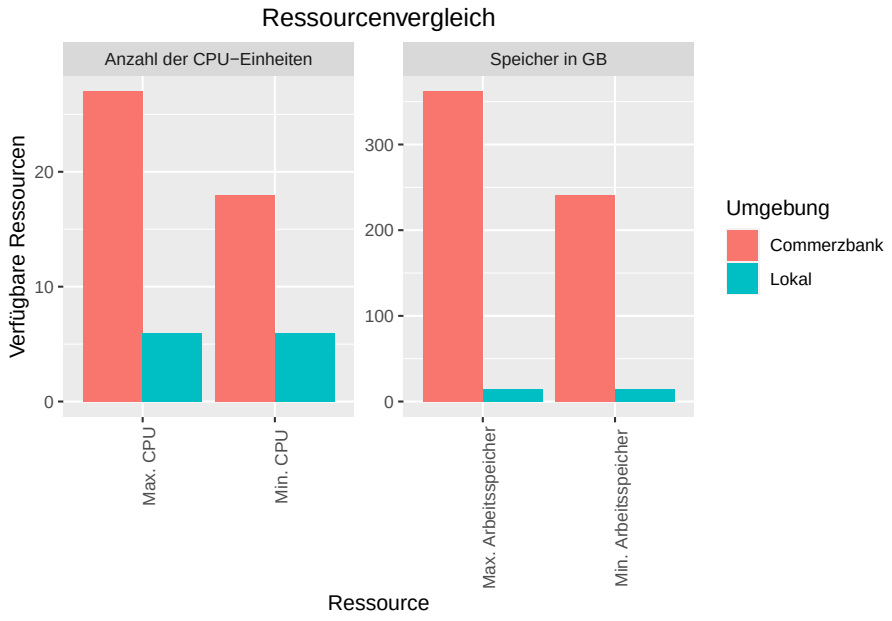


Abbildung 5.3: Vergleich der verfügbaren Ressourcen

5.3 TESTMETRIKEN

Im folgenden Abschnitt werden die in der Evaluation verwendeten Testmetriken vorgestellt:

- Die Laufzeiten werden in Sekunden gemessen. Die Zeitmessung wird dann gestartet, wenn alle notwendigen Ressourcen verfügbar sind und mit der Berechnung begonnen werden kann. Die Laufzeit der Verfahren Domänenberechnung, Ähnlichkeitsberechnung und Verbundberechnung werden separat

erfasst. So kann ermittelt werden, auf welche Verfahren die Laufzeit aufzuteilen ist.
- Neben der Laufzeit wird die Suchraumreduktion durch gewählte Optimierungsstrategien untersucht. Hierfür wird die Suchraumreduktion in Prozent bestimmt.
- Es wird untersucht, wann ein Ergebnis True-Positive (TP) oder False-Positive (FP) ist. Mit der Konfusionsmatrix aus Tabelle 5.5 wird dies anhand von relevanten und nicht relevanten Ergebnissen eines Klassifikators verdeutlicht.

	Relevant	**Nicht relevant**
Positives Ergebnis	TP: Zwischen zwei Attributen besteht **eine** Beziehung und diese ist **erkannt** worden.	FP: Zwischen zwei Attributen besteht **keine** Beziehung und diese wird **nicht** erkannt.
Negatives Ergebnis	TN: Zwischen zwei Attributen besteht **keine** Beziehung und dies ist **erkannt** worden.	FN: Zwischen zwei Attributen besteht **eine** Beziehung und diese wird **nicht** erkannt.

Tabelle 5.5: Konfusionsmatrix nach [7][vgl. 192]

5.4 TESTFALLBESCHREIBUNG

Die Testfälle sind aus den Anforderungen aus dem Kapitel 2 abzuleiten. Aus den Anforderungen ergeben sich acht Testfälle:

Testfall 1

- **Testfallbeschreibung:** Die Performance des Schema-Evolutions-Prozesses ist nicht abhängig von der Menge der zu betrachteten Zeilen in einem Datensatz. Getestet werden das TPC-H-Datenmodell mit einer Datenmenge von 2, 3 und 5 GB an Daten und das Datenmodell der Commerzbank mit jeweils 1 (114 GB), 3 (344 GB) und 5 (574 GB) Tagesbelieferungen[2]. Unterschieden werden die Teilaspekte: Domänenklassifikation, Ähnlichkeitsberechnung und Verbundberechnung.
- **Testmetrik:** Laufzeitmessung in Sekunden
- **Testerfolg:** Wenn die Laufzeit nicht überlinear ansteigt, gilt der Testfall als erfolgreich.
- **Testmisserfolg:** Wenn ein überlineares Laufzeitverhalten verzeichnet wird, gilt der Testfall als nicht erfolgreich.

Testfall 2

- **Testfallbeschreibung:** Es soll getestet werden, welche Auswirkungen größere Datenmodelle auf das Laufzeitverhalten haben. Da das TPC-H-Datenmodell nicht erweitert werden kann, wird das Datenmodell der Commerzbank um weitere Entitäten ergänzt. Es werden 5 Tageslieferungen betrachtet mit jeweils 5, 15 und 50 Entitäten. Dieser Testfall kann nur im Kontext der Commerzbank ausgeführt werden.
- **Testmetrik:** Laufzeitmessung in Sekunden
- **Testerfolg:** Der Testfall gilt als erfolgreich, wenn eine lineare Steigerung der Laufzeit zu verzeichnen ist.

2 Eine Tagesbelieferung umfasst alle Daten, die innerhalb eines Banktages in dem jeweiligen Datenmodell erfasst worden sind.

- **Testmisserfolg:** Da die Anzahl der Entitäten pro Testschritt mehr als verdoppelt werden, ist eine maximale Verdopplung der Laufzeit noch zu tolerieren. Längere Laufzeiten müssen untersucht werden und führen zum Nichtbestehen des Testfalles.

Testfall 3

- **Testfallbeschreibung:** Kann mit einer Ressourcenerhöhung das Laufzeitverhalten verbessert werden? Da das Datenmodell der Commerzbank nicht exportiert werden darf, wird das TPC-H-Datenmodell für diesen Testfall verwendet. Es werden jeweils $1, 3, 5$ und 25 GB an Datensätzen erzeugt und evaluiert.
- **Testmetrik:** Laufzeitmessung in Sekunden
- **Testerfolg:** Der Testfall gilt als erfolgreich, wenn eine annähernd lineare Laufzeitsteigerung durch eine Ressourcenerhöhung verzeichnet werden kann.
- **Testmisserfolg:** Falls eine überlineare Laufzeitsteigerung erreicht wird, ist dies ein Indiz für eine schlechte Skalierung der Verfahren. In einem solchen Fall gilt der Testfall als nicht erfolgreich.

Testfall 4

- **Testfallbeschreibung:** Es muss sichergestellt werden, dass alle Testobjekte vollständig berücksichtigt werden. Dies gilt besonders für die Berechnung der Domänen, der Ähnlichkeiten und der Verbünde.
- **Testmetrik:** Abdeckung in Prozent
- **Testerfolg:** Der Testfall gilt als erfolgreich, wenn alle Testobjekte berücksichtigt werden. Wenn ein Testobjekt ausgelassen wird, muss dies gemäß Spezifikation begründbar sein.
- **Testmisserfolg:** Wenn nur Teilmengen der Testobjekte berücksichtigt werden und dieses Verhalten nicht begründet werden kann, gilt der Testfall als nicht erfolgreich.

Testfall 5

- **Testfallbeschreibung:** Kann der Suchraum durch die gewählten Optimierungsstrategien reduziert werden?
- **Testmetrik:** Reduzierung in Prozent
- **Testerfolg:** Die Reduzierung des Suchraums ist eine Grundvoraussetzung für einen praxisnahen Einsatz der Verfahren. Ein Suchraum wird durch die Anzahl der zu überprüfenden Attributspaare definiert. Als maximale Laufzeit der Verfahren Ähnlichkeitsberechnung und Verbundberechnung werden 48 Stunden festgelegt. Die Verarbeitung eines Attributspaares benötigt zwei Sekunden. Erst wenn durch das Anwenden der Optimierungsstrategien der Suchraum dermaßen reduziert werden kann, dass eine Verarbeitung innerhalb von 48 Stunden möglich ist, gilt der Testfall als erfolgreich.
- **Testmisserfolg:** Auch durch das Anwenden der Optimierungsstrategien die maximale Laufzeitgrenze von 48 Stunden nicht eingehalten werden kann, gilt der Testfall als nicht erfolgreich.

Testfall 6

- **Testfallbeschreibung:** Es ist zu überprüfen, ob die ermittelten Ähnlichkeiten und Verbünde korrekt sind. Die Überprüfung kann anhand der Spezifikation

der Datenmodelle erfolgen oder alternativ anhand einer manuellen Datensichtung.
- **Testmetrik:** Manuelle Beurteilung
- **Testerfolg:** Der Testfall gilt als erfolgreich, wenn FPs nicht vorhanden sind oder diese erklärt werden können.
- **Testmisserfolg:** Wenn FPs vorhanden sind und diese nicht erklärt werden können, gilt der Testfall als nicht erfolgreich.

Testfall 7

- **Testfallbeschreibung:** Können Schema-Operation aus den Daten extrahiert werden? Die Multitype-Schema-Operationen *Move* und *Copy* werden durch eine Schema-Veränderung in den Entitäten erzeugt und müssen anschließend ermittelt werden.
- **Testmetrik:** Manuelle Beurteilung
- **Testerfolg:** Der Testfall gilt als erfolgreich, wenn die Multitype-Schema-Operationen korrekt erkannt werden.
- **Testmisserfolg:** Wenn die Multitype-Schema-Operationen nicht oder unvollständig extrahiert werden können, gilt der Testfall als nicht erfolgreich.

Testfall 8

- **Testfallbeschreibung:** Können mithilfe des Pattern-Verbundes Beziehungen zwischen Attributen gefunden werden, deren alphabetische Attributswerte unterschiedlich skaliert sind?
- **Testmetrik:** Manuelle Beurteilung
- **Testerfolg:** Der Testfall gilt als erfolgreich, wenn für alphabetischen Attributen Patterns für einen Verbund gefunden werden können.
- **Testmisserfolg:** Wenn keine Patterns gefunden werden, gilt der Testfall als nicht erfolgreich.

5.5 TESTDURCHFÜHRUNG

Testfall 1

Ziel dieses Testfalles ist es, herauszufinden, ob ein praxistauglicher Einsatz der vorgestellten Verfahren möglich ist. Durch eine Variation des Datenvolumens wird untersucht, ob eine vertikale Skalierung, bezogen auf die Menge an Daten in „realistischer“[3] Zeit verarbeitet werden kann. Das Datenmodell ist für diesen Test statisch. Durch eine mehrfache Ausführung und Mittelung der Laufzeiten können Schwankungen der erhaltenen Ressourcen ausgeglichen werden.

Die Testobjekte beider Datenmodelle sind in Tabelle 5.1 aufgelistet. Für das Datenmodell der Commerzbank stehen fünf Stichtage (574 GB) drei Stichtagen (344 GB) und eine Stichtag (114 GB) zur verfügung. Die Testdaten für das TPC-H-Datenmodell werden mehrmals mit unterschiedlichem Datenvolumen für diesen Testfall generiert.

Die Laufzeiten werden Verfahren einzeln erhoben. So ergeben sich Laufzeiten für das Berechnen der Domänen, der Ähnlichkeiten und der Verbünde. In den Abbildungen 5.4 und 5.5 ist eine lineare Laufzeitsteigerung zu erkennen. Eine Verdoppelung des Datenvolumens hat eine Verdopplung der Laufzeit zur Folge.

3 Eine Verarbeitung ist nicht mehr realistisch, wenn mehr als zwei Tage für eine Ausführung benötigt werden. Dieser Grenzwert ergibt sich daraus, dass langlaufende Verarbeitungen über das Wochenende durchgeführt werden können und in diesem Zeitraum keine Ressourcen für Ad-hoc-Anfragen blockiert werden können.

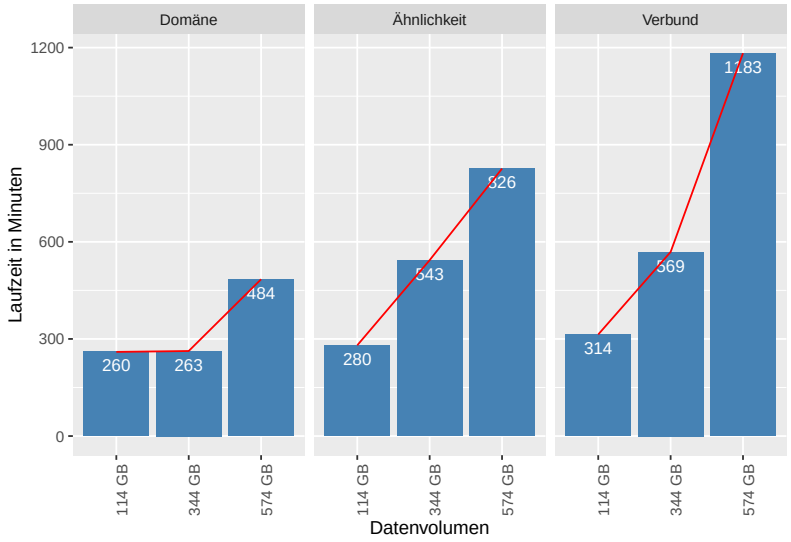


Abbildung 5.4: Laufzeitverhalten bei vertikaler Datenskalierung: Datenmodell Commerzbank

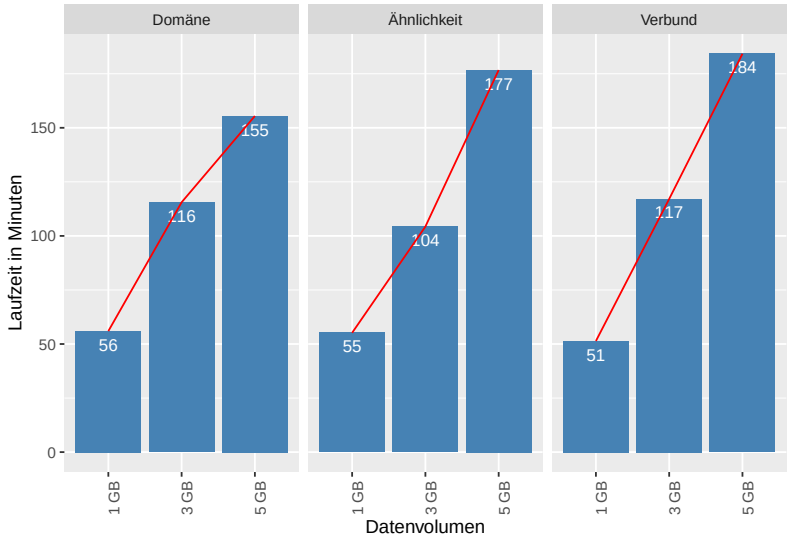


Abbildung 5.5: Laufzeitverhalten bei vertikaler Datenskalierung: Datenmodell TPC-H

Diese Laufzeiten sind einmalige Investitionen des Schema-Evolutions-Prozesses. Die Verfahren müssen einmalig vollständig und anschließend inkrementell berechnet werden. Die unbefriedigende[4] Laufzeitsteigerung lässt sich durch die Art der Stichprobenerzeugung erklären:

Eine Entität wird basierend auf einem ausgewiesenen Attribut in Buckets unterteilt. Die Zuweisung eines Datensatzes zu einem Bucket wird durch das Anwenden einer Hash-Funktion definiert. Eine Anfrage kann auf einzelne Elemente innerhalb eines Buckets beschränkt werden. Im Stichprobenerzeugungsprozess wird die Anzahl der Buckets, die für eine Stichprobe verwendet werden sollen, definiert. In Abhängigkeit der Datenverteilung kann die Anzahl der Datensätze für eine Stichprobe auch in den jeweiligen Buckets hoch sein. Wenn ein Attribut einen Datensatz nicht

4 Durch das Erzeugen von Stichproben wird eine nicht ansteigende Laufzeit bei einer Variation des zu verarbeitenden Datenvolumens erwartet.

in gleich große Buckets unterteilen kann, ist die Stichprobengröße abhängig von der Größe des Datenvolumens. Dies kann nicht getestet werden, da weder im TPC-H-Datenmodell noch im Commerzbank-Datenmodell ein Attribut gefunden werden kann, das die Bedingung an die Gleichverteilung eines Datensatzes erfüllen kann. Die Idee wird im folgenden Testdatensatz demonstriert. Als Hash-Funktion wird Modulo 2 verwendet.

X	Hash-Funktion	Bucket
1	1 mod 2	1
2	2 mod 2	0
3	3 mod 2	1
4	4 mod 2	0
5	5 mod 2	1
6	6 mod 2	0

Tabelle 5.6: Gleichverteilung von Elementen in Buckets

Y	Hash-Funktion	Bucket
1	1 mod 2	1
1	1 mod 2	1
1	1 mod 2	1
1	1 mod 2	1
1	1 mod 2	1
2	2 mod 2	0

Tabelle 5.7: Ungleiche Verteilung von Elementen in Buckets

In Tabelle 5.6 ergibt sich nach dem Anwenden der Hash-Funktion auf das Attribut X eine Gleichverteilung der Elemente in den Buckets. Hingegen ist in Tabelle 5.7 zu sehen, dass eine überproportionale Anzahl an Elementen in einem Bucket versammelt sind. Dies deutet darauf hin, dass das gewählte Attribut Y nicht für eine Gleichverteilung der Attributswerte verwendet werden kann [30][vgl. 422]. Wenn im Stichprobenerzeugungsprozess die Buckets aus der Tabelle 5.7 Verwendung finden, ist keine Reduzierung des Stichprobenvolumens zu verzeichnen und eine lineare Laufzeitsteigerung ist die Folge.

Testfall 2

Im ersten Testfall wurde die vertikale Datenskalierung getestet. Zusätzlich muss die horizontale Entitätsskalierung untersucht werden. Hierfür wird das Datenmodell der Commerzbank um weitere Entitäten erweitert. In Tabelle 5.8 sind die unterschiedlichen Entitätsskalierungen gezeigt.

Anzahl Entitäten	Anzahl Datenbanken	Anzahl Attribute
5	2	78
15	2	357
50	2	803

Tabelle 5.8: Schemavolumen

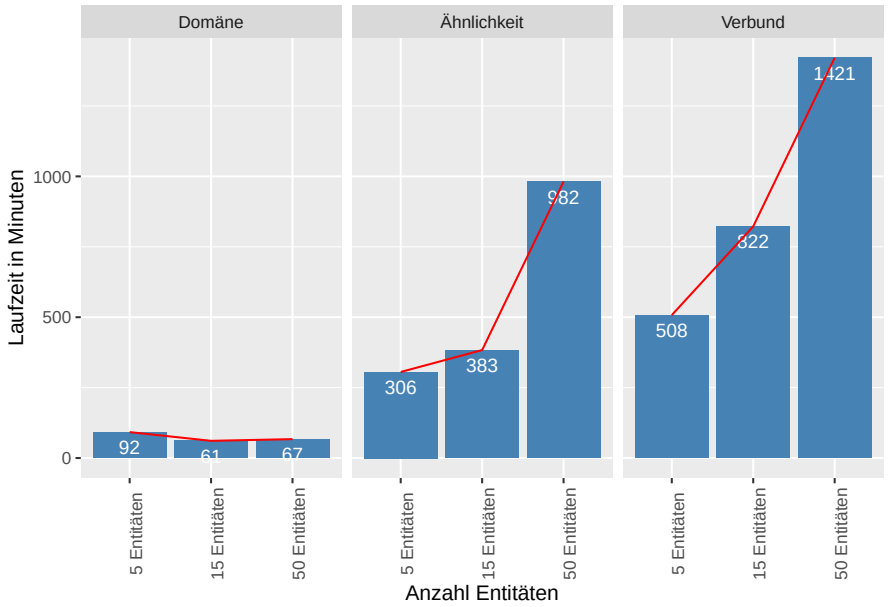


Abbildung 5.6: Laufzeitverhalten bei horizontaler Skalierung

In Abbildung 5.6 ist zu sehen, dass die Anzahl der Attribute keinen Einfluss auf das Laufzeitverhalten der Domänenberechnung hat. Zwischen den Testfällen wird die Anzahl der Attribute mehr als verdreifacht. Die Laufzeit wird verdoppelt, wenn die Anzahl der Attribute verdreifacht wird. In Anbetracht der praxisnahen Anzahl der Entitäten ist diese Laufzeitsteigerung günstig zu bewerten.

Testfall 3

Im Folgenden wird getestet, welchen Einfluss zusätzliche Ressourcen auf das Laufzeitverhalten der Verfahren haben. Da das Datenmodell der Commerzbank nicht in der lokalen Testumgebung evaluiert werden darf, wird stattdessen das TPC-H-Datenmodell in der Testumgebung der Commerzbank implementiert und evaluiert.

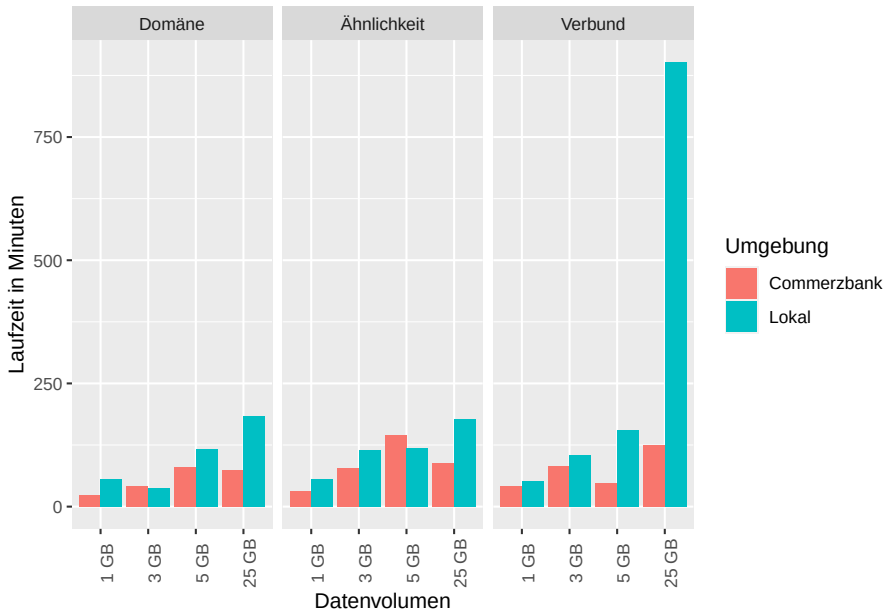


Abbildung 5.7: Ressourcenabhängiges Laufzeitverhalten

In Abbildung 5.7 zeigt sich, dass durch eine Ressourcenerhöhung die Laufzeiten reduziert werden können. Wichtig ist der Ausschlag der Datenverarbeitung von $25GB$ in der lokalen Testumgebung. Da der verfügbare Arbeitsspeicher auf $15GB$ der lokalen Testumgebung limitiert ist, ergibt sich eine überproportionale Laufzeitsteigerung. Dies könnte daran liegen, dass die Ergebnisse der Datenverarbeitung nicht durchgängig im Arbeitsspeicher gehalten werden können und auf die Festplatte gespeichert werden müssen [13][vgl. 307]. Somit ist eine Ressourcenerhöhung nicht ausschließlich für eine Laufzeitsteigerung notwendig, sondern auch für die Verarbeitung von hohen Datenmengen.

Testfall 4

Die Verarbeitung der Testobjekte muss vollständig erfolgen, da im Schema-Evolutions-Prozess jegliche Schema-Veränderungen erkannt werden müssen. Die Vollständigkeit kann durch einen Abgleich mit der Anzahl der verfügbaren Attribute im Vergleich zu der Anzahl der betrachteten Attribute überprüft werden. In Tabelle 5.9 ist die Abdeckung pro Verfahren und Datenmodell aufgelistet.

Umgebung	**Verfahren**	**Abdeckung**
Commerzbank	Domäne	$\frac{443}{443} = 1$
Commerzbank	Ähnlichkeit	$\frac{206}{443} = 0.465$
Commerzbank	Verbund	$\frac{126}{443} = 0.284$
TPC-H	Domäne	$\frac{60}{60} = 1$
TPC-H	Ähnlichkeit	$\frac{55}{60} = 0.916$
TPC-H	Verbund	$\frac{30}{60} = 0.5$

Tabelle 5.9: Datenvollständigkeitsevaluation

Es zeigt sich, dass alle Attribute für die Domänenberechnung betrachtet werden. Eine Abdeckung von 100% ist für beide Datenmodelle zu verzeichnen. Es lassen sich Abweichungen in der Ähnlichkeits- und Verbundberechnung beobachten. Diese können durch die Optimierungsstrategien aus Kapitel 3.3.2 erklärt werden. Alle Attributspaare, die nicht zueinander kompatibel sind, werden der Ähnlichkeits- und Verbundberechnung nicht zugeführt. Auch ein Ähnlichkeitswert wird nicht bestimmt. Diese Attribute haben keine Beziehung zueinander und sind nicht kompatibel. Für das TPC-H-Datenmodell sind folgende Attribute für die Ähnlichkeitsberechnung nicht berücksichtigt worden:

Entität	**Attribut**	**Domäne**
lineitem	l_linestatus	None
nation	n_comment	None
nation	n_name	None
nation	n_nationkey	None
nation	n_regionkey	None

Tabelle 5.10: Datenvollständigkeit bezüglich der Ähnlichkeitsberechnung

In Tabelle 5.10 sind ausschließlich Attribute in der Domänenklasse None aufgelistet. Diese Attribute werden von der Ähnlichkeitsberechnung ausgeschlossen. Wenn

diese Attribute der Gesamtanzahl der betrachteten Attribute hinzugefügt werden, ergibt sich eine Abdeckung von $\frac{55+5}{60} = 1 = 100\%$.

Entität	**Attribut**	**Domäne**
customer	c_address	Alphabetisch
customer	c_comment	Alphabetisch
customer	c_name	Alphabetisch
customer	c_phone	Alphabetisch
lineitem	l_comment	Alphabetisch
lineitem	l_commitdate	Alphabetisch
lineitem	l_discount	Metrisch
lineitem	l_linestatus	Dichotom
lineitem	l_quantity	Metrisch
lineitem	l_receiptdate	Alphabetisch
lineitem	l_shipdate	Alphabetisch
lineitem	l_tax	Metrisch
nation	n_comment	None
nation	n_name	None
nation	n_nationkey	None
nation	n_regionkey	None
orders	o_clerk	Alphabetisch
orders	o_comment	Alphabetisch
orders	o_orderdate	Alphabetisch
part	p_comment	Alphabetisch
part	p_container	Alphabetisch
part	p_name	Alphabetisch
part	p_type	Alphabetisch
partsupp	ps_comment	Alphabetisch
revenue	total_revenue	Metrisch
supplier	s_acctbal	Metrisch
supplier	s_address	Alphabetisch
supplier	s_comment	Alphabetisch
supplier	s_name	Alphabetisch
supplier	s_phone	Alphabetisch

Tabelle 5.11: Datenvollständigkeit bezüglich der Verbundberechnung

In Tabelle 5.11 sind die nicht betrachteten Attribute für die Verbundberechnung aufgelistet. Anhand der Verbundkompatibilitätsmatrix ist nachvollziehbar, warum diese Attribute nicht betrachtet wurden. Durch die Hinzunahme dieser Attribute ergibt sich eine Datenvollständigkeit von $\frac{30+30}{60} = 1 = 100\%$. Für die Vollständigkeitsberechnung des Datenmodells der Commerzbank kann durch die Hinzunahme der bewusst ausgeschlossenen Attribute für allen Verfahren ein Vollständigkeitswert von 100% erreicht werden.

Testfall 5

Optimierungsstrategien sind ein wichtiger Baustein in der Verarbeitung von großen Datenmengen. Die Anzahl der ausgeschlossenen Attributspaare sind in Tabelle 5.12 je Optimierungsstrategie gezeigt.

Umgebung	**Verfahren**	**Optimierungsstrategie**	**Startgröße**	**Endgröße**	**Reduktion um**
Commerzbank	Ähnlichkeit	Domänenkompatibilität	93.528	40.382	0,431
Commerzbank	Ähnlichkeit	Anwendungsbereich	40.382	36.436	0,17
Commerzbank	Ähnlichkeit	Kommutativität	36.436	18.218	0,5
Commerzbank	Ähnlichkeit	Transitivität	18.218	18.218	0
Commerzbank	Verbund	Domänenkompatibilität	93.528	9.660	0,103
Commerzbank	Verbund	Anwendungsbereich	9.660	9.064	0,061
Commerzbank	Verbund	Kommutativität	9.064	4.532	0,5
Commerzbank	Verbund	Transitivität	4.532	4.532	0
Commerzbank	Verbund	Statistiken	4.532	1.858	0,419
TPC-H	Ähnlichkeit	Domänenkompatibilität	2.080	924	0,444
TPC-H	Ähnlichkeit	Anwendungsbereich	924	748	0.144
TPC-H	Ähnlichkeit	Kommutativität	748	374	0.5
TPC-H	Ähnlichkeit	Transitivität	374	374	0
TPC-H	Verbund	Domänenkompatibilität	2.080	527	0,253
TPC-H	Verbund	Anwendungsbereich	527	448	0,153
TPC-H	Verbund	Kommutativität	448	224	0.5
TPC-H	Verbund	Transitivität	224	224	0
TPC-H	Verbund	Statistiken	224	224	0

Tabelle 5.12: Zusammenfassung der Optimierungsstrategien

Es zeigt sich, dass mit Ausnahme der Optimierungsstrategie *Transitivität* eine Reduzierung des Suchraums erreicht werden kann. Eine transitive Beziehung kann nur ausgenutzt werden, wenn transitive Ähnlichkeits- oder Verbundbeziehungen zwischen Attributen in den Datensätzen vorhanden sind. Dies ist weder im TPC-H-Datenmodell noch im Datenmodell der Commerzbank der Fall.

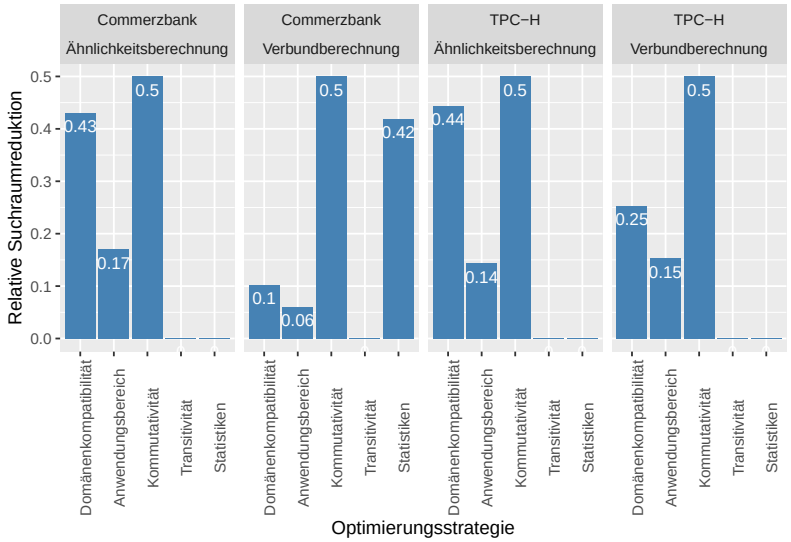


Abbildung 5.8: Gegenüberstellung der Optimierungsstrategien

Abbildung 5.8 zeigt die Suchraumreduktion der verschiedenen Optimierungsstrategien. Es fällt auf, dass in der Ähnlichkeitsberechnung für beide Datenmodelle durch die *Kommutativität* eine deutliche Suchraumreduktion erreicht werden kann. In der Verbundberechnung ist die Betrachtung der Statistiken für das Datenmodell der Commerzbank sinnvoll. Da das TPC-H-Datenmodell künstlich erzeugt wurde, können die Statistiken den Suchraum nicht reduzieren. Es ist zu beachten, dass die Optimierungsstrategien in diesem Testfall sukzessive den Suchraum für die Ähnlichkeits- und Verbundberechnung reduzieren. Die Optimierungsstrategien arbeiten mit demjenigen Suchraum, der von den vorangegangenen Optimierungsstrategien verblieben ist. Wenn eine vorangegangene Optimierungsstrategie den Suchraum so reduziert hat, dass keine weiteren Optimierungen möglich sind, führt dies dazu, dass einige Optimierungsstrategien anscheinend keine Reduzierung des Suchraums aufweisen. Da die Optimierungsstrategien alleinig auf den verfügbaren Metadaten basieren, ist jegliche Reduzierung des Suchraums sinnvoll und sollte durchgeführt werden.

Die maximale Verarbeitungslaufzeit wird auf zwei Tage festgelegt, da eine Verarbeitung innerhalb eines Wochenendes (ca. 48 Stunden) abgeschlossen sein muss. In dieser Zeit werden keine potenziell businesskritischen Ad-hoc-Anfragen gestellt und können durch die Verarbeitung der Verfahren dieser Arbeit nicht blockiert werden. Ein Attributspaar kann in zwei Sekunden verarbeitet werden. Die Verarbeitungszeit kann in der Testumgebung der Commerzbank beobachtet werden.

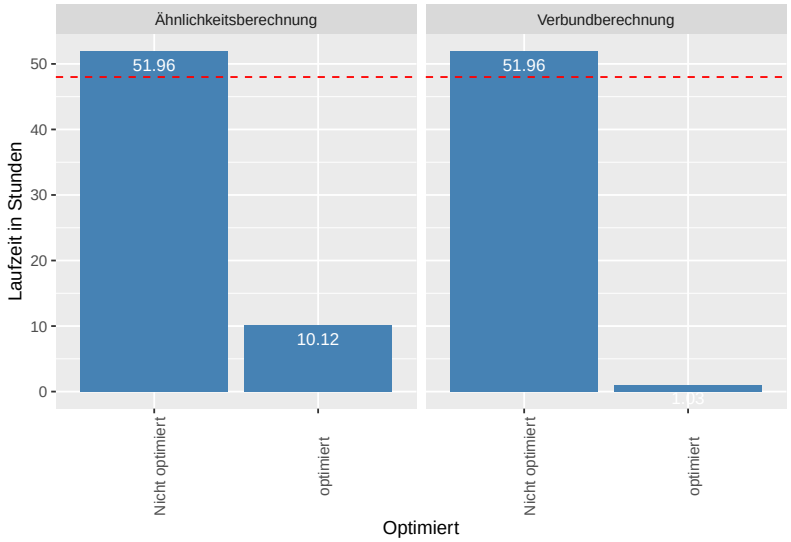


Abbildung 5.9: Mögliche Verarbeitung im Kontext der Commerzbank

In Abbildung 5.9 sind die Gesamtlaufzeiten für die Ähnlichkeits- und Verbundberechnung für die Testumgebung der Commerzbank zu sehen. Ohne die Optimierungsstrategien ist ein praxistauglicher Einsatz der Verfahren nicht möglich. Es ist zu beachten, dass die Laufzeit von zwei Sekunden ein empirisch ermittelter Wert ist, der in Abhängigkeit der Ressourcen und des Datenmodells variieren kann.

Zusammenfassend ergibt sich für die Ähnlichkeitsberechnung eine Suchraumreduktion von 80,52% für das Datenmodell der Commerzbank und 82,01% für das TPC-H-Datenmodell. Für die Verbundberechnung kann eine Suchraumreduktion von 98,01% für das Datenmodell der Commerzbank und 89,23% für das TPC-H-Datenmodell erreicht werden.

Testfall 6

Die Korrektheit der ermittelten Ergebnisse ist ein wichtiger Baustein für einen praxistauglichen Einsatz der Verfahren. Weiterhin werden die Verfahren Ähnlichkeits- und Verbundberechnung einzeln betrachtet. In Tabelle 5.13 ist eine Stichprobe von Attributspaaren zu sehen, die entweder in einer Ähnlichkeits- oder Verbundbeziehung stehen. Die Ergebnisse können anhand der Spezifikation der Datenmodelle oder anhand einer manuellen Datensichtung überprüft werden.

ID	Umgebung	Verfahren	Quelle			Ziel			Korrekt
			Entität	Attribut	Domäne	Entität	Attribut	Domäne	
1	Commerzbank	Ähnlichkeit	exp_party	bpkenn	UCC	exp_verbund	verbund_kopf	UCC	FP
2	Commerzbank	Ähnlichkeit	exp_abweichende_gpkdnr	kdnr	UCC	exp_gp	fuehrende_kdnr	UCC	TP
3	Commerzbank	Ähnlichkeit	exp_party_legi_orga	bpkenn	UCC	exp_gp_struktur	gp_objekt	UCC	TP
4	Commerzbank	Ähnlichkeit	exp_gp	gpkenn	UCC	exp_party	bpkenn	UCC	TP
5	Commerzbank	Ähnlichkeit	exp_gp	gpkenn	UCC	exp_party_legi_person	bpkenn	UCC	TP
6	Commerzbank	Ähnlichkeit	exp_abweichende_gpkdnr	kdnr	UCC	exp_struktur	kdnr	UCC	TP
7	Commerzbank	Ähnlichkeit	exp_gp	gpkenn	UCC	exp_verbund	vbkenn	Ordinal	TP
8	Commerzbank	Verbund	exp_kdnr	kdnr	UCC	exp_gp	fuehrende_kdnr	UCC	TP 5.14
9	Commerzbank	Verbund	exp_party	gpkenn_tech	UCC	exp_gp	gpkenn	UCC	TP 5.14
10	Commerzbank	Verbund	exp_struktur	gpkenn_fach	UCC	exp_gp	gpkenn	UCC	TP 5.14
11	Commerzbank	Verbund	exp_gp	fuehrende_kdnr	UCC	exp_kdnr	kdnr	UCC	TP 5.14
12	Commerzbank	Verbund	exp_party_adresse_p	postleitzahl	UCC	exp_party	wz_branche	Ordinal	FP 5.14
13	Commerzbank	Verbund	exp_party_adresse_p	postleitzahl	UCC	exp_party_adresse_e	anschluss	Ordinal	FP 5.14
14	Commerzbank	Verbund	exp_verbund_struktur	gp_objekt	UCC	exp_verbund	verbund_kopf	UCC	TP 5.14
15	TPC-H	Ähnlichkeit	lineitem	l_partkey	UCC	part	p_partkey	UCC	TP
16	TPC-H	Ähnlichkeit	lineitem	l_shipdate	Alphabetisch	orders	o_orderdate	Alphabetisch	TP
17	TPC-H	Ähnlichkeit	customer	c_comment	Alphabetisch	lineitem	l_comment	Alphabetisch	TP
18	TPC-H	Ähnlichkeit	supplier	s_comment	Alphabetisch	customer	c_comment	Alphabetisch	TP
19	TPC-H	Ähnlichkeit	orders	o_orderdate	Alphabetisch	lineitem	l_receiptdate	Alphabetisch	TP
20	TPC-H	Verbund	orders	o_custkey	Ordinal	customer	c_custkey	UCC	TP 5.14
21	TPC-H	Verbund	orders	o_orderkey	Ordinal	supplier	s_suppkey	UCC	FP 5.14
22	TPC-H	Verbund	orders	o_orderkey	Ordinal	customer	c_custkey	UCC	TP 5.14
23	TPC-H	Verbund	part	p_partkey	UCC	partsupp	ps_partkey	Ordinal	TP 5.14
24	TPC-H	Verbund	part	p_partkey	UCC	lineitem	l_partkey	Ordinal	FP 5.14
25	TPC-H	Verbund	supplier	s_suppkey	UCC	orders	o_orderkey	Ordinal	FP 5.14
26	TPC-H	Verbund	supplier	s_suppkey	UCC	partsupp	ps_availqty	Ordinal	FP 5.14
27	TPC-H	Verbund	supplier	s_suppkey	UCC	partsupp	ps_suppkey	Ordinal	TP 5.14
28	TPC-H	Verbund	supplier	s_suppkey	UCC	lineitem	l_suppkey	Ordinal	FP 5.14

Tabelle 5.13: Zusammenfassung der ermittelten Ähnlichkeiten und Verbünde

Für die Evaluation der Verbünde ist in Tabelle 5.14 die Anzahl der Zeilen gelistet, die durch einen Verbund zwischen X und Y entstehen. Ein Attributspaar besteht aus einem Quell-Attribut X und einem Ziel-Attribut Y. Das Verhältnis der Anzahl der Verbundwerte zur Gesamtanzahl der Attributswerte ist ein Indikator für die Beziehung zwischen X und Y. Die Anzahl der Attributswerte des Verbundes kann durch 1:n-Beziehungen größer sein als die Gesamtanzahl der Attributswerte.

ID	$\lvert X \ltimes Y \rvert$	$\lvert X \rvert + \lvert Y \rvert$	$\frac{\lvert X \ltimes Y \rvert}{\lvert X \rvert + \lvert Y \rvert}$
8	768.592	14.723.663	0,0522
9	23.621.199	23.621.199	1
10	16.844.382	16.844.382	1
11	768.592	768.699	0,99
12	4.179.600.048	17.785.782	234,99
13	2.479.449.405	17.785.782	139,4
14	1.334.490	634.197	2,1
20	750.000	750.000	1
21	1.255	750.000	0,001
22	18.751	750.000	0,02
23	400.000	100.000	4
24	2.999.671	100.000	29,99
25	1.255	5.000	0,25
26	20.040	5.000	4
27	400.000	5.000	80
28	2.999.671	5.000	599,93

Tabelle 5.14: Evaluierung der Verbünde

Für eine abschließende Bewertung der Verfahren wird eine Konfusionsmatrix berechnet:

	Commerzbank-Datenmodell	**TPC-H-Datenmodell**
TP	6	5
FP	1	0

Tabelle 5.15: Konfusionsmatrix für die Ähnlichkeitsberechnung

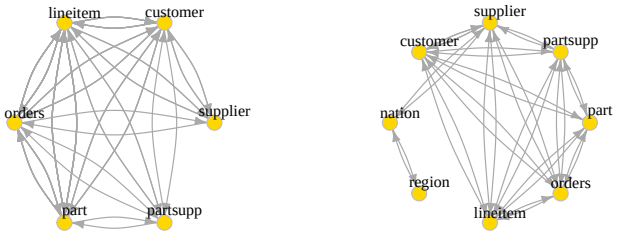
	Commerzbank-Datenmodell	**TPC-H-Datenmodell**
TP	5	2
FP	2	7

Tabelle 5.16: Konfusionsmatrix für die Verbundberechnung

In Tabelle 5.15 und 5.16 ist die Konfusionsmatrix für die Ähnlichkeits- und Verbundberechnung zu sehen. Ausprägungen für FN und TN können nicht bestimmt werden: Ein FN ist aufgetreten, wenn Ähnlichkeits- oder Verbundbeziehungen in den Datensätzen vorliegen, diese aber nicht erkannt werden. Ein TN hingegen beschreibt die Anzahl aller erkannten nicht-existenten Ähnlichkeits- oder Verbundbeziehungen. Da diese Mengen entgegen der Annahme nicht vollständig spezifiziert

sind und die Menge der zu vergleichenden Attributspaare zu hoch ist, werden stattdessen die Ähnlichkeits- und Verbundbeziehungen für eine Beurteilung der Vollständigkeit visualisiert.

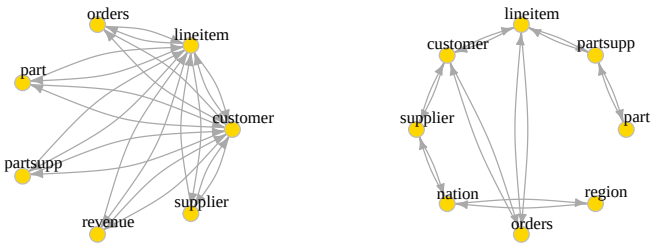
Ein Graph wird durch eine Knoten- und Kantenmenge definiert. Die Knotenmenge umfasst alle Entitäten eines Datenmodells. Eine Verbindung zwischen zwei Knoten wird hinzugefügt, wenn zwischen zwei Attributen unterschiedlicher Entitäten eine Ähnlichkeits- oder Verbundbeziehung besteht. So kann übersichtlich nachvollzogen werden, ob zwischen Entitäten Ähnlichkeiten oder Verbünde fehlen. Die Vollständigkeit wird nur eingeschränkt überprüft, da ein einzelnes Attribut ausreicht, um zwischen zwei Entitäten eine Kante zu erzeugen. Da die Anzahl der ähnlichen Attribute pro Entität klein ist, wird diese Problematik akzeptiert. In Abbildung 5.10 sind alle Ähnlichkeitsbeziehungen zwischen den Entitäten des TPC-H-Datenmodells als gerichteten Graph dargestellt.



Beobachtete Ähnlichkeitsbeziehungen Erwartete Ähnlichkeitsbeziehungen

Abbildung 5.10: Ähnlichkeitsgraph des TCP-H-Datenmodells

Es fällt auf, dass für die Entitäten *nation* und *region* keine Ähnlichkeiten ermittelt werden können. Da für diese Entitäten nur wenige Datensätze verfügbar sind, können keine Ähnlichkeiten berechnet werden. Anhand des Graphen kann festgestellt werden, dass wesentliche Kanten vorhanden sind. Dies spricht für eine gute Abdeckung der ermittelten Ähnlichkeiten. Selbige Verfahrensweise wird für die Analyse der Verbünde durchgeführt:



Beobachtete Verbundbeziehungen Erwartete Verbundbeziehungen

Abbildung 5.11: Verbundgraph des TCP-H-Datenmodells

Die Entitäten *nation* und *region* sind in der Menge der ermittelten Verbünde nicht vertreten, da zu wenige Datensätze in diesen Entitäten vorhanden sind. Es zeigt sich, dass die Graphstrukturen deutlich voneinander abweichen. So ist eine Beziehung zwischen der Entität *customer* und *orders* in beiden Graphen vorhanden. Zu-

sätzlich wird eine Beziehung zwischen *customer* und *part* ermittelt. Dieses Verhalten lässt sich mit dem Generierungsprozess der Testdaten begründen. Da für alle Primär- und Fremdschlüssel die jeweils gleichen Initialisierungswerte verwendet werden, existieren zusätzliche Verbünde zwischen Entitäten, die nicht spezifiziert sind. Für das Datenmodell der Commerzbank werden ausschließlich die beobachteten Ähnlichkeits- und Verbundbeziehungen gezeigt. Das erwartete Datenmodell darf nicht veröffentlicht werden.

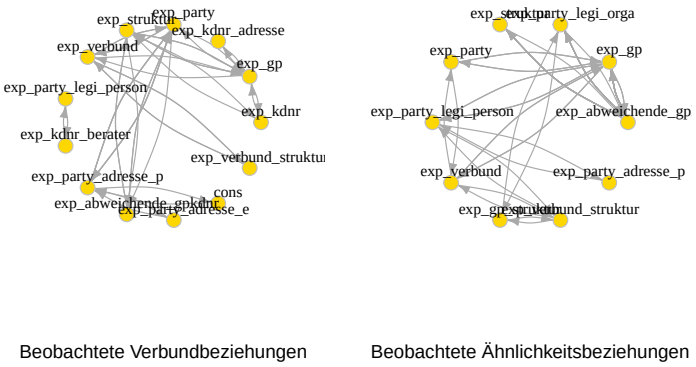


Abbildung 5.12: Ähnlichkeits- und Verbundgraph des Commerzbank-Datenmodells

Die beobachteten Ähnlichkeits- und Verbundbeziehungen decken sich mit dem erwarteten Datenmodell.

Testfall 7

Die Ermittlung von Multitype-Schema-Operationen ist ein allumfassender Test der Verfahren. Es werden je drei Multitype-Schema-Operationen pro Datenmodell durchgeführt. Diese müssen anschließend aus den Daten extrahiert werden. In Tabelle 5.17 sind die durchgeführten und ermittelten Schema-Operationen aufgelistet.

ID	Umgebung	Operation	Durchgeführte (D) und extrahierte (E) Multitype-Schema-Operation
1	Commerzbank	Verschieben	(D) **MOVE** exp_gp.gp_status to exp_gp_organkredit.**status** WHERE exp_gp.**gpkenn** = exp_gp_organkredit.**gpkenn** (E) **Move** exp_gp_v1.**gp_status** TO exp_gp_organkredit_v2.**gp_status** WHERE lhs.**gpkenn** = rhs.**gpkenn**
2	Commerzbank	Kopieren	(D) **COPY** exp_kdnr_adresse.**strasse** to exp_kdnr.**str** WHERE exp_kdnr.**kdnr** = exp_kdnr_adresse.**kdnr** (E) **Copy** exp_kdnr_adresse_v1.**strasse** TO exp_kdnr_v2.**str** WHERE lhs.**kdnr** = rhs.**kdnr**
3	Commerzbank	Kopieren	(D) **COPY** exp_party_adresse_p.**strasse** to exp_party_adresse_e.**str** WHERE exp_party_adresse_e.**bpkenn** = exp_party_adresse_p.**bpkenn** (E) **Copy** exp_party_adresse_e_v2.**str** TO exp_party_adresse_p_v1.**strasse** WHERE lhs.**bpkenn** = rhs.**bpkenn**
4	TPC-H	Verschieben	(D) **MOVE** order.**o_comment** to customer.**comment** WHERE order.**o_custkey** = customer.**c_custkey** (E) **Move** orders_v1.**o_comment** TO $customer_v2$.**comment** WHERE lhs.**o_custkey** = rhs.**c_custkey**
5	TPC-H	Kopieren	(D) **COPY** part.**p_name** to partsupp.**name** WHERE part.**p_partkey** = partsupp.**ps_partkey** (E) **Copy** part_v1.**p_name** TO partsupp_v2.**name** WHERE lhs.**p_size** = rhs.**ps_availqty**
6	TPC-H	Kopieren	(D) **COPY** orders.**o_orderpriority** to supplier.**prio** WHERE orders.**o_orderkey** = supplier.**s_suppkey** (E) **Copy** orders_v1.**o_orderpriority** TO supplier_2.**prio** WHERE lhs.**o_custkey** = rhs.**s_suppkey**

Tabelle 5.17: Evaluierung der Multitype-Schema-Operationen

Die erzeugten Multitype-Schema-Operationen konnten in beiden Datenmodellen zuverlässig erkannt werden. Im Testfall 3 sind die LHS und RHS Attribute vertauscht. Dies deutet darauf hin, dass in deren Ausführung die Quell und Ziel Entität verwechselt worden sind. Da das Starten des Schema-Evolutions-Prozesses ein manueller Schritt[5] ist, ist dieser Fehler hierauf zurückzuführen. Der Testfall 5 hingegen weist eine von der Spezifikation abweichende Verbundbeziehung zwischen den Entitäten auf. Nach einer Datensichtung konnte festgestellt werden, dass die gewählte Verbundbeziehung ein FP ist:

Zwischen den Entitäten *part* und *partsupp* mit dem Verbund *part.p_size = partsupp.ps_availqty* können 399.453.250 Tupel gefunden werden. Wenn stattdessen der spezifizierte Verbund aus *part.p_partkey = partsupp.ps_partkey* verwendet wird, werden 4.000.000 Tupel erzeugt. Die Verfahren agieren gemäß der Spezifikation. Dieser FP zeigt, dass die Semantik und die Validität eines Verbundes nicht alleinig durch die Tupelanzahl ermittelt werden kann. Da dieser Fall in den bisherigen Testfällen nicht aufgetreten ist, wird dieser Missstand akzeptiert.

Testfall 8

Neben des IND-Verbundes kann ein Entitätsverbund auch anhand eines Patterns berechnet werden. Eine konsolidierte Sicht auf den Verbund zweier Entitäten wird durch das Verbinden der einzelnen Patterns erreicht. In Tabelle 5.18 wird auf einer Testdatenbank gezeigt, welche Möglichkeiten durch einen Pattern-Verbund erreicht werden können.

Quelle		**Ziel**		**Key**	**Value**
Entität	**Attribut**	**Entität**	**Attribut**		
$customer_v1$	address	$customer_v2$	addr	Antioquia	Antioquia
$customer_v1$	address	$customer_v2$	addr	Iowa	Iowa
$customer_v1$	address	$customer_v2$	addr	KPK	KPK
$customer_v1$	address	$customer_v2$	addr	Surrey	Surrey
$customer_v1$	lastname	$customer_v2$	lname	Amery	Amery
$customer_v1$	lastname	$customer_v2$	lname	Emery	Amery
$customer_v1$	lastname	$customer_v2$	lname	Jolie	Jolie
$customer_v1$	fname	$customer_v2$	firstname	Tana	Cara
$customer_v1$	fname	$customer_v2$	firstname	Eve	Eve
$customer_v1$	fname	$customer_v2$	firstname	Karly	Karen
$customer_v1$	fname	$customer_v2$	firstname	Karly	Karly
$customer_v1$	fname	$customer_v2$	firstname	Karly	Kelly

Tabelle 5.18: Evaluation des Pattern-Verbundes

Neben sehr unauffälligen Pattern, zum Beispiel bei Gleichheit zweier Attributswerte, lassen sich auch Pattern finden, die für eine weitere Datenverarbeitung interessant sind. Für den Schema-Evolutions-Prozess findet der Pattern-Verbund für das Verbinden von alphabetischen Attributen Anwendung.

5 In der Evaluation ist das Starten des Schema-Evolutions-Prozesses ein manueller Schritt.

5.6 TESTPARAMETER

Der Parameter θ wird als Schwellwert für die Klassifizierung zweier Attribute in die Klassen *ähnlich* und *disjunkt* beziehungsweise in die Klassen *verbunden* und *disjunkt* verwendet. Für die Berechnung der Ähnlichkeits- und Verbundwerte wird dieser Parameter nicht berücksichtigt. Anhand der Ähnlichkeitswert- und Verbundwertverteilung aus Abbildung 5.13 können passende Schwellwerte für θ gefunden werden.

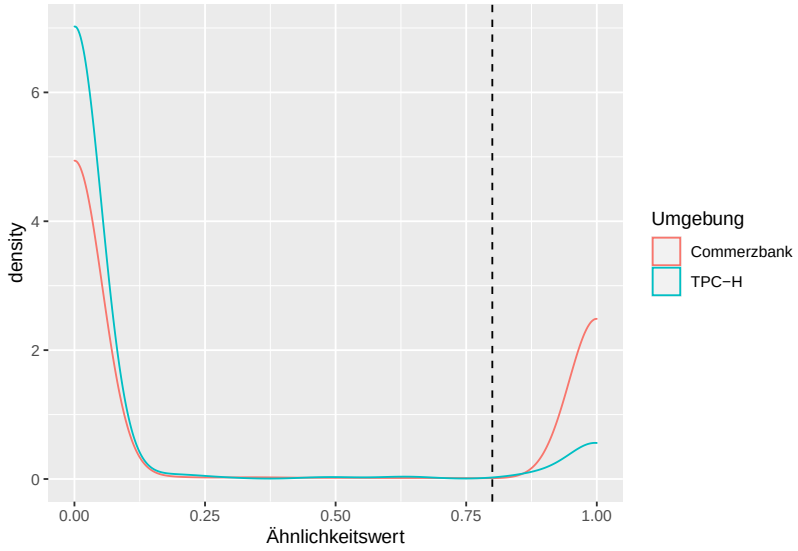


Abbildung 5.13: Ähnlichkeitswertverteilung

In Abbildung 5.13 ist die Ähnlichkeitswertverteilung beider Datenmodelle zu sehen. Ab einem Ähnlichkeitswert von ungefähr 0.8 ist eine Ballung in der Verteilung zu sehen. Ab dieser Grenze werden Attributspaare der Klasse *ähnlich* zugewiesen. Selbige Überlegung wird in Abbildung 5.14 für die Verbundwerte durchgeführt:

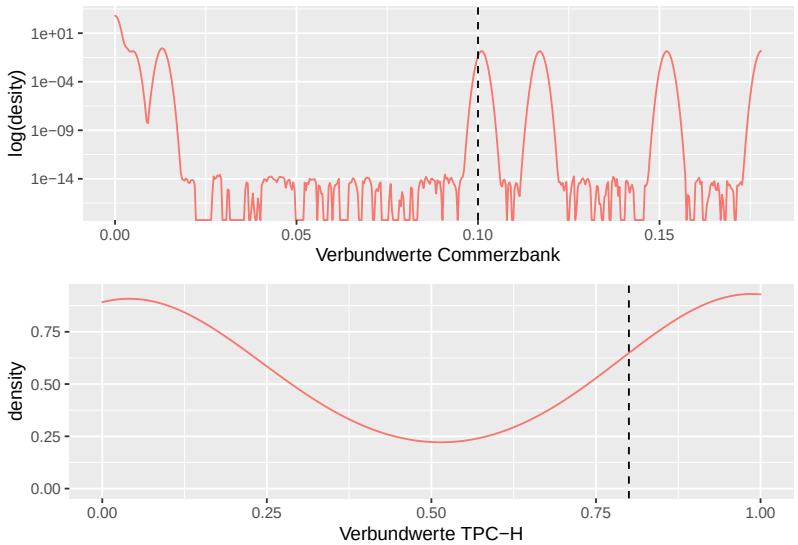


Abbildung 5.14: Verbundwertverteilung

Die Verbundwertverteilung ist zwischen den Datenmodellen verschieden. Da das TPC-H-Datenmodell künstlich erzeugt wurde, sind alle Verbundoperationen fehler-

frei und erhalten hohe Verbundwerte. Im Datenmodell der Commerzbank sind nur wenige Stichtage für die Berechnung der Verbundwerte verfügbar. Kleine Verbundwerte werden zwischen zwei Attributen (X und Y) erzeugt, wenn die Menge aus $|X \subseteq Y| \lll |X|$ ist. Der Schwellwertparameter für den Verbund des Datenmodells der Commerzbank wird aus diesem Grund sehr niedrig gewählt. In Tabelle 5.19 sind alle Schwellwertparameter aufgelistet:

Umgebung	Ähnlichkeitsberechnung	Verbundberechnung
Commerzbank	0.8	0.1
TPC-H	0.8	0.8

Tabelle 5.19: Testparameter

5.7 TESTBERICHT

Im Testbericht wird die Testdurchführung Revue passiert und gemäß der definierten Erfolgs- und Misserfolgs-Kriterien bewertet.

Testfall-Nummer	Begründung	Wertung
1	Die Laufzeit verhält sich linear mit einer Erhöhung der Anzahl an Datensätzen. Dieses Verhalten ist aufgrund der Datenbeschaffenheit begründbar und wird akzeptiert.	👉
2	Für die horizontale Skalierung ist eine lineare Laufzeitsteigerung zu verzeichnen. Dies gilt nach der Definition des Testfalles als Erfolg.	👍
3	Durch eine Ressourcenerhöhung kann eine Laufzeitreduktion erreicht werden. Dies ist für kleine Datenmengen nicht signifikant. Der Testfall gilt somit als nur teils erfolgreich.	👉
4	Die vollständige Datenverarbeitung konnte für beide Datenmodelle nachgewiesen werden. Nicht berücksichtigte Attribute konnten am Beispiel des TPC-H-Datenmodells erläutert werden. Der Testfall gilt als erfolgreich.	👍
5	Der Suchraum konnte durch die gewählten Optimierungsstrategien reduziert werden. Eine Verarbeitung ist nur mit den beschriebenen Optimierungsstrategien praxistauglich, das heißt, eine Verarbeitung kann innerhalb von 48 Stunden abgeschlossen werden.	👍
6	Die ermittelten FP der Ähnlichkeits- und Verbundberechnungen können erklärt werden. Der Testfall ist erfolgreich.	👍
7	Alle Multitype-Schema-Operationen konnten festgestellt werden. Der Testfall ist erfolgreich.	👍
8	Es zeigt sich, dass der Pattern-Verbund eine sinnvolle Ergänzung zu den bestehenden IND-Verbunden ist. Der Testfall gilt als erfolgreich.	👍

5.8 WEITERE ERKENNTNISSE

Neben den beschriebenen Testfällen sind weitere Erkenntnisse aus den Daten zu erheben.

Schema-Historie

Das Thema Schema-Evolution ist kein theoretisches Konstrukt. Dies zeigt auch die Abbildung 5.15 deutlich. Innerhalb eines Monats wurden in einer Testumgebung der Commerzbank bis zu 2.418 Schema-Veränderungen registriert. Diese hohe Anzahl motiviert die Notwendigkeit nach einem automatisierten Schema-Evolutions-Prozess. Jede Schema-Veränderung wie Erstellung, Veränderung oder Entfernung einer Entität muss protokolliert werden. Diese Informationen dienen dem Nachvollziehen der Schema-Historie.

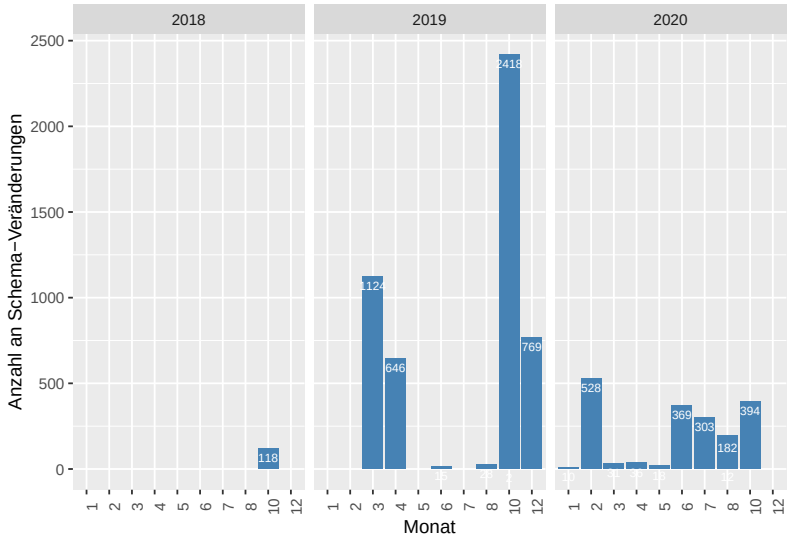


Abbildung 5.15: Anzahl an Schema-Veränderungen der Jahre 2018-2020

Das für einige Monate keine Schema-Veränderungen zu verzeichnen ist (vgl. Abbildung 5.15) liegt daran, dass die zu untersuchende Testumgebung im Oktober 2018 den Betrieb aufgenommen hat. Weitere Datenlücken lassen sich durch Urlaube begründen.

Ähnlichkeitsmaßverteilung

Die verwendeten Ähnlichkeitsmaße sind abhängig von der Domänenverteilung der Datenmodelle. In Abbildung 5.16 ist ersichtlich, dass die relativen Häufigkeiten der verwendeten Ähnlichkeitsmaße beider Datenmodelle verschieden sind. Die überproportionale Verwendung des JIN-Tests im Datenmodell der Commerzbank ist auf eine Überrepräsentation der Domänenklasse ordinal zurückzuführen. Ebenso ist ein Unterschied im KS-Test zu erkennen. Dies kann durch eine hohe Anzahl an dichotomen Attributen im Datenmodell der Commerzbank erklärt werden.

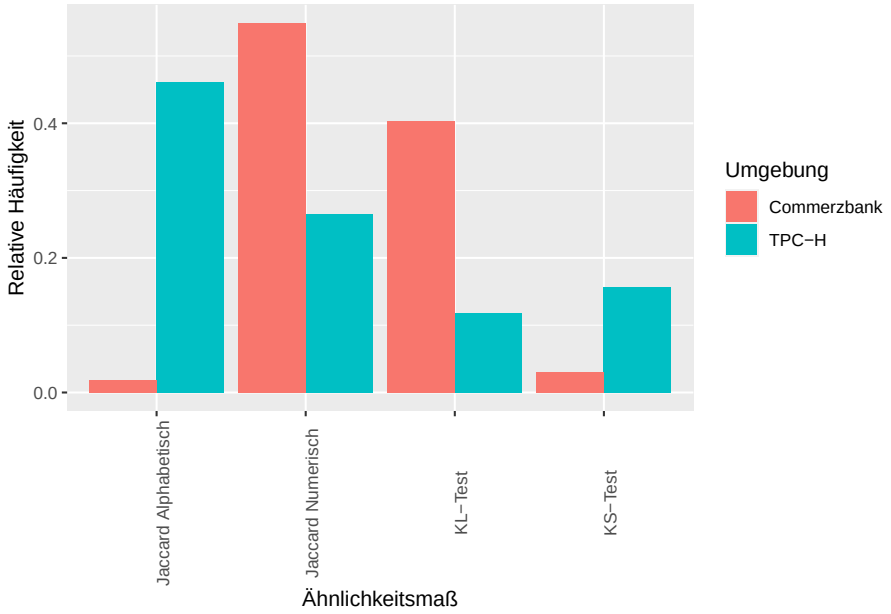


Abbildung 5.16: Verteilung der verwendeten Ähnlichkeitsmaße

Domänenverteilung

Die konkreten Verteilungen der Domänen sind in Abbildung 5.17 zu finden. Die hohe Anzahl der dichotomen und None-Domänen lässt sich durch die Beschaffenheit der Testdaten erklären. Wenn innerhalb eines Stichtages maximal zwei Ausprägungen für ein Attribut vorliegen, wird ein Attribut der dichotomen Domänenklasse zugewiesen. Zusätzlich kann innerhalb eines Stichtages nur eine begrenzte Anzahl an Attributswerten für vermeintlich alphabetische Domänen vorhanden sein. Dies erklärt die hohe Anzahl an ordinalen und kategorialen Domänen.

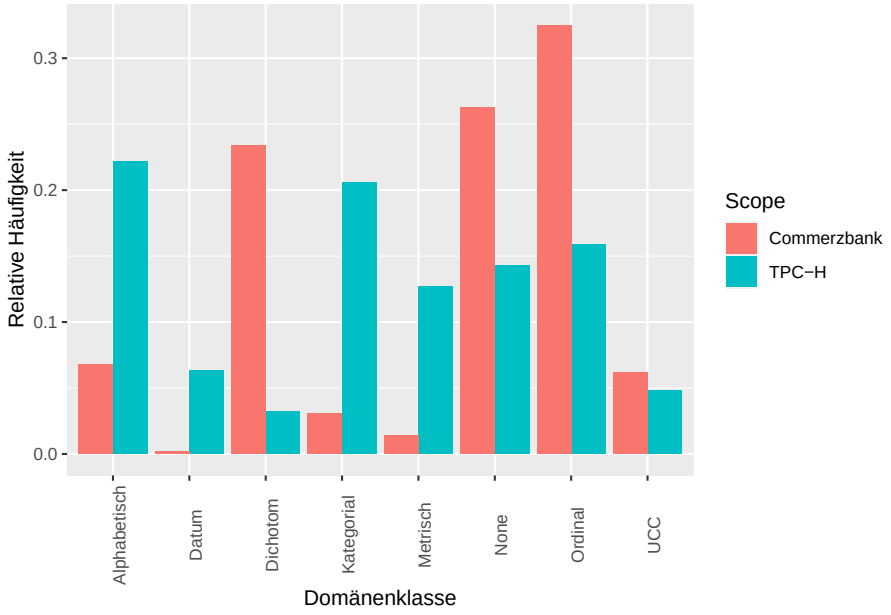


Abbildung 5.17: Verteilung der Domänenklassen

Da eine Domäne eine Abstraktion eines Datentyps ist, sind in Tabelle 5.20 die definierten Datentypen der Entitäten im Commerzbank-Datenmodell kumuliert gezeigt. Da mehre Datentypen auf eine Domäne verweisen können, wird zusätzlich deren Zuweisung definiert.

Datentyp	Relative Häufigkeit	Zugehörige Domäne
boolean	0,0005	Dichotom
smallint	0,0077	Metrisch
tinyint	0,0188	Metrisch
int	0,0379	Metrisch
bigint	0,0015	Metrisch
decimal	0,2669	Metrisch
double	0,0031	Metrisch
date	0,0081	Datum
timestamp	0,0863	Datum
string	0,0108	Alphabetisch
varchar	0,5579	Alphabetisch

Tabelle 5.20: Datentypverteilung des Commerzbank-Datenmodells

In Abbildung 5.18 ist die Domänenverteilung der echten Datentypen gezeigt. Es zeigt sich eine Überrepräsentation der alphabetischen und metrischen Domänen. Dies liegt daran, dass anhand des Datentyps nicht entschieden werden kann, ob ein Attribut kategorial, ordinal, dichotom oder UCC ist. So wird den Attributen mithilfe der Domänen eine weitere semantische Ebene hinzugefügt.

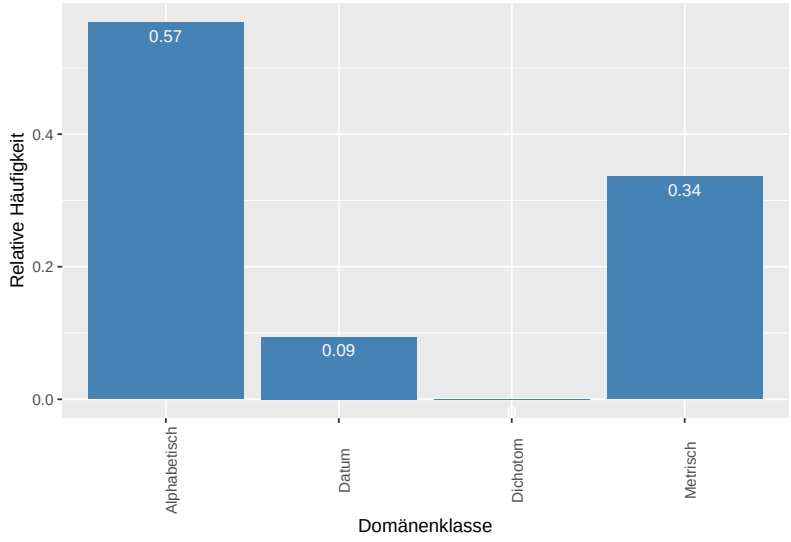


Abbildung 5.18: Verteilung der Domänenklassen (Abgeleitet von dem Datentyp)

Domänenpaarungen

Neben der Definition der Kompatibilitätsmatrix für die Betrachtung eines Attributspaares für die Ähnlichkeits- oder Verbundberechnung ist auch die Domänenverteilung interessant. In Abbildung 5.19 ist für beide Datenmodelle die relative

Häufigkeitsverteilung der möglichen Domänenpaarungen der Ähnlichkeitsmaße gezeigt. Dunklere Farben visualisieren einen höheren relativen Häufigkeitswert einer Domänenpaarung. Die Verteilungen der beiden Datenmodelle lassen sich nicht miteinander vergleichen, da jeweils unterschiedlich verteilte Domänenklassen zugrunde liegen. Auffällig ist jedoch, dass das Datenmodell der Commerzbank zu einer spärlich besetzten Matrix führt. Da eine Überrepräsentation der ordinalen Domäne in den Daten zu verzeichnen ist, ist diese Attributkombination auch in der Berechnung eines Ähnlichkeitswertes mit fast 50% vertreten. Das TPC-H-Datenmodell ist hingegen weniger konzentriert auf eine Domänenpaarung und somit ist die daraus resultierende Matrix dichter besetzt.

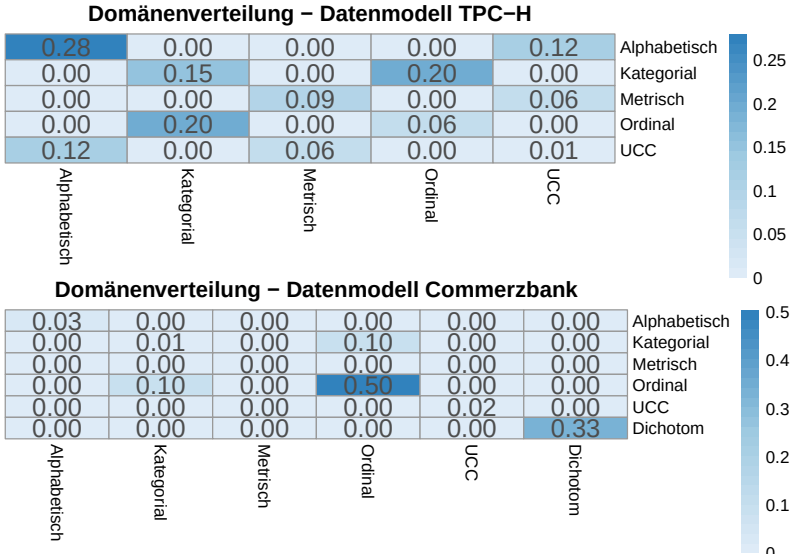


Abbildung 5.19: Verteilung der verwendeten Ähnlichkeitsmaßkombinationen

Selbige Überlegung kann für die Betrachtung der Domänenverteilung für den Verbund erzeugt werden. Abbildung 5.20 zeigt, dass in beiden Datenmodellen die Kombinationen UCC und ordinal mit Abstand am häufigsten für die Verbundberechnung Verwendung finden. Eine UCC ist eine notwendige Bedingung für einen Primärschlüssel. Ein ordinales Attribut kann einen Fremdschlüssel zu einer UCC bilden.

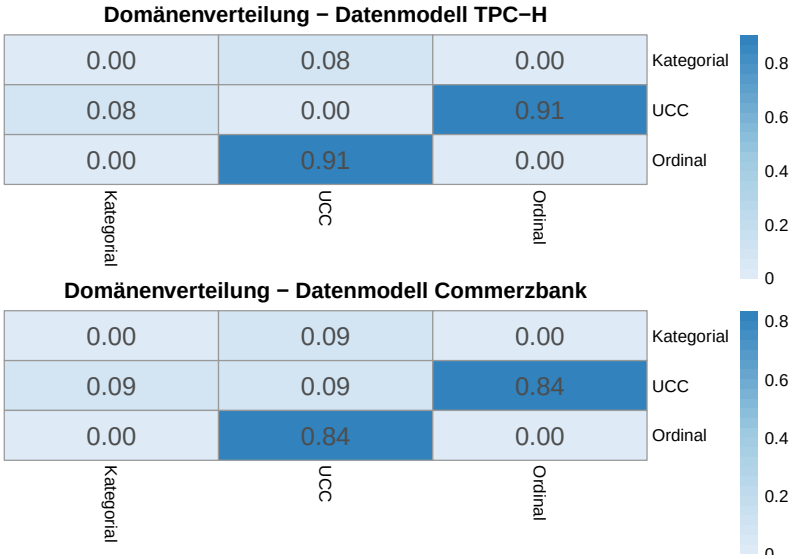


Abbildung 5.20: Verteilung der verwendeten Verbundkombinationen

6 FAZIT UND AUSBLICK

FAZIT

In der Evaluation konnte gezeigt werden, dass die zu Beginn gestellten Anforderungen nachweislich erfüllt werden können. Durch die Hinzunahme eines realistischen Datenmodells und dessen Evaluation ist ein Schritt hin zu einer produktiv verwendbaren Anwendung gelegt worden. Das Ziel, die Multitype-Schema-Operationen *Move* und *Copy* datengetrieben zu erkennen, konnte nachweislich erfüllt werden. Die Verfahren in dieser Arbeit sind nicht proprietär auf eine konkrete Implementierung eines Datenbankherstellers zugeschnitten. Somit besteht die Möglichkeit, mit geringem Aufwand ein Datenbanksystem gegen ein anderes auszutauschen.

Das Thema Schema-Evolution ist ein bedeutendes Thema für die Commerzbank. Alle Schema-Veränderungen müssen protokolliert und nachvollziehbar gemacht werden. Datenkonsumenten erhalten somit eine stabile Grundlage für eine erfolgreiche Zusammenarbeit.

Eine Arbeit im Bereich Evolution ist unvollständig, ohne Charles Darwin zu erwähnen:

> „It is not the most intellectual of the species that survives; it is not the strongest that survives; but the species that survives is the one that is able to adapt to and to adjust best to the changing environment in which it finds itself.“[9]

Angepasst auf den Kontext der Schema-Evolution bedeutet dies, dass nicht die intelligentesten Systeme und Datenmodelle überdauern werden. Alleinig die Flexibilität auf Veränderungen zu reagieren und sich kontinuierlich neu zu erfinden, ist die Erfolgsgeschichte der natürlichen und technischen Evolution.

AUSBLICK

6.0.1 *Singletype-Schema-Operationen*

Der Fokus dieser Arbeit lag in der Erkennung der Multitype-Schema-Operationen *Move* und *Copy*. Die Singletype-Schema-Operationen *Add*, *Delete* und *Rename* können durch eine Erweiterung abgedeckt werden: In Abbildung 6.1 ist der Entscheidungsbaum für die Unterscheidung von Schema-Operationen zu sehen.

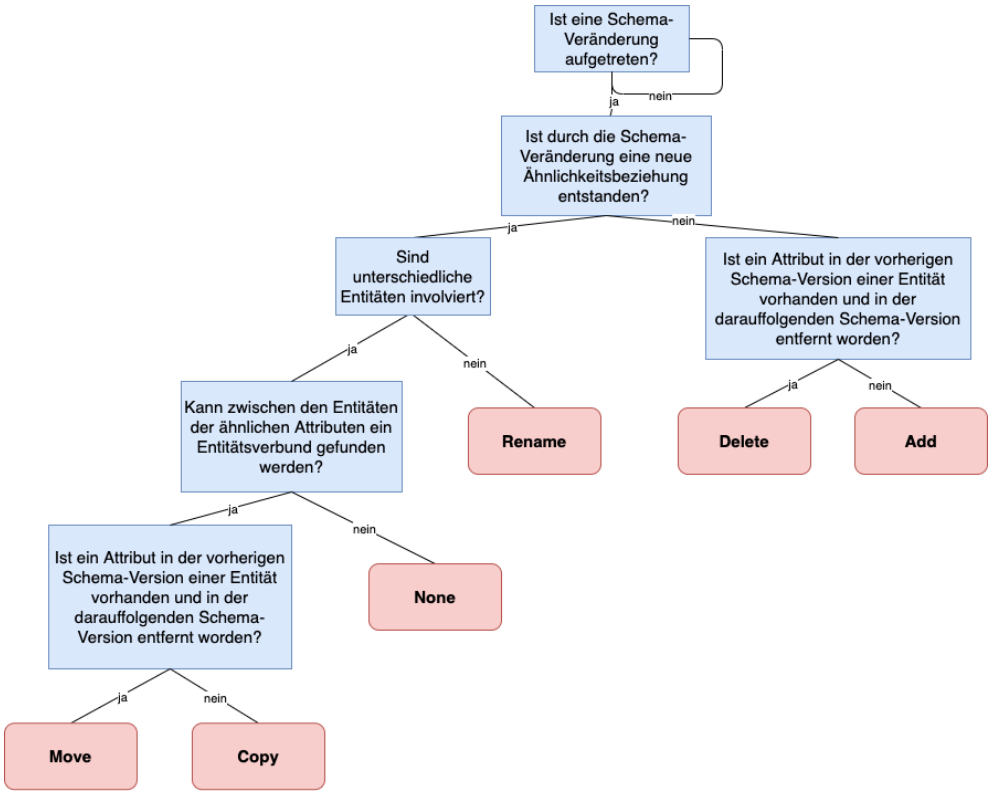


Abbildung 6.1: Entscheidungsbaum für Schema-Operationen

Im Folgenden werden die Singletype-Schema-Operationen behandelt: Eine Schema-Veränderung wurde erkannt. Anschließend wird die Differenzmenge zwischen der aktuellen und der vorherigen Schema-Version der veränderten Entität berechnet. Wenn ein Attribut dieser Menge zu einem anderen Attribut eine Ähnlichkeitsbeziehung aufweist, kann es sich um eine *Move*-, *Copy*- oder *Rename*-Schema-Operation handeln. Eine *Rename*-Schema-Operation findet innerhalb einer Entität statt und setzt eine Ähnlichkeitsbeziehung zwischen zwei Attributen voraus. Wenn eine Ähnlichkeitsbeziehung zwischen zwei Attributen besteht, deren Entitäten verschieden sind, kann eine Multitype-Schema-Operation aufgetreten sein. Das Erkennen von Multitype-Schema-Operationen ist in Kapitel 3 nachzulesen. Die Schema-Operationen *Add* und *Delete* können durch eine Analyse der Schema-Versionen isoliert werden. Wenn ein Attribut in einer vorherigen Schema-Version einer Entität vorhanden ist und in der darauffolgenden Schema-Version entfernt wurde, dann ist die Schema-Operation *Delete* aufgetreten. Fehlt hingegen ein Attribut in einer vorherigen Schema-Version und wird in einer darauffolgenden Schema-Version hinzugefügt, ist die Schema-Operation *Add* aufgetreten. Singletype-Schema-Operationen sind im Vergleich zu Multitype-Schema-Operationen weniger aufwendig, da die Verbundberechnung entfällt.

Es zeigt sich, dass durch eine Anpassung des Entscheidungsbaumes Multitype- und Singletype-Schema-Operationen erkannt werden können.

6.0.2 *Datenbereinigung*

Wenn ein Attribut mehr als nur eine Fachlichkeit abdeckt, ist die Atomarität eines Attributs gebrochen. Angenommen ein Attribut a ist in der Schema-Version S_{v0} definiert. Die Werte in a sind nicht atomar. Es gibt nun eine Schema-Veränderung in der Datenmodellierung, die a betreffen. In der Datenmodellierung wird die Atomarität von a durch ein Aufteilen auf die Attribute b und c gebrochen und mit der Schema-Version S_{v1} in eine Datenbank eingebracht. Durch die Verwendung von Ähnlichkeitsmaßen kann im Schema-Evolutions-Prozess festgestellt werden, dass zwischen den beiden Schema-Versionen $player_{v0}$ und $player_{v1}$ die Attribute $a \sim b$ und $a \sim c$ sind. Mithilfe der Armstrong-Axiome [38, vgl. 2] kann nun geschlossen werden, dass $a \sim \{b, c\}$ gilt. Dies wird anhand eines Beispiels deutlich:

In Tabelle 6.1 ist die initiale Schema-Version einer vereinfacht dargestellten *Player*-Entität zu finden. Das Attribut *name* beinhaltet, durch ein Komma getrennt, den Vor- und Nachnamen von Spielern. In Tabelle 6.2 ist dieses Attribut auf die Attribute *vorname* und *nachname* aufgeteilt worden.

ID	**name**
1	Dominik, Ludwig
2	Millou, Hermes
3	Millow, Amazon
4	Fabian, Ludwig
5	Marie, Schubert
6	Christine, Ludwig

Tabelle 6.1: Player-Entität Schema-Version v_0

ID	**name**	**nachname**
1	Dominik	Ludwig
2	Millou	Hermes
3	Millow	Amazon
4	Fabian	Ludwig
5	Marie	Schubert
6	Christine	Ludwig

Tabelle 6.2: Player-Entität Schema-Version v_1

Es kann im Schema-Evolutions-Prozess erkannt werden, dass $name \sim vorname$ und $name \sim nachname$ gilt. Daraus folgt $name \sim \{vorname, nachname\}$. Für eine Daten-Migration der Schema-Versionen $player_{v0}$ zu $player_{v1}$ muss jenes Pattern gefunden werden, das beschreibt, wie die Attributswerte aufgeteilt werden. In dem Beispiel ist dies ein Komma. Komplexere Aufteilungen lassen sich mithilfe eines regulären Ausdruckes beschreiben. Dem Schema-Evolutions-Prozess muss mitgeteilt werden, wie die einzelnen Attribute zusammengeführt werden können. Auch hier ist die Definition eines Patterns notwendig. Das Ermitteln eines Patterns stellt eine noch zu lösende Herausforderung dar. Ansätze können in [22] gefunden werden. Anhand der Daten wird ein regulärer Ausdruck berechnet. Dieser kann für diese split/join-Operation verwendet werden.

Teil II

APPENDIX

A

KRITISCHE WERTE DES KS-TESTS

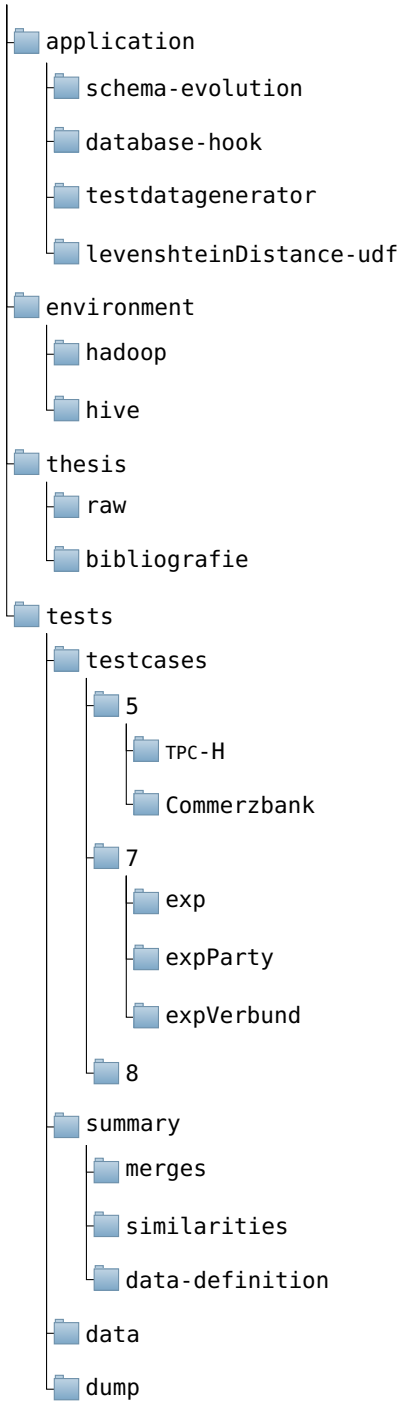
n	0.1	0.05	0.025	0.01	0.005
1	0.9000	0.9500	0.9750	0.9900	0.9950
2	0.6838	0.7764	0.8419	0.9000	0.9293
3	0.5648	0.6360	0.7076	0.7846	0.8290
4	0.4927	0.5652	0.6239	0.6889	0.7342
5	0.4470	0.5094	0.5633	0.6272	0.6685
6	0.4104	0.4680	0.5193	0.5774	0.6166
7	0.3815	0.4361	0.4834	0.5384	0.5758
8	0.3583	0.4096	0.4543	0.5065	0.5418
9	0.3391	0.3875	0.4300	0.4796	0.5133
10	0.3226	0.3687	0.4092	0.4566	0.4889
11	0.3083	0.3524	0.3912	0.4367	0.4677
12	0.2958	0.3382	0.3754	0.4192	0.4490
13	0.2847	0.3255	0.3614	0.4036	0.4325
14	0.2748	0.3142	0.3489	0.3897	0.4176
15	0.2659	0.3040	0.3376	0.3771	0.4042
16	0.2578	0.2947	0.3273	0.3657	0.3920
17	0.2504	0.2863	0.3180	0.3553	0.3809
18	0.2436	0.2785	0.3094	0.3457	0.3706
19	0.2373	0.2714	0.3014	0.3369	0.3612
20	0.2316	0.2647	0.2941	0.3287	0.3524
21	0.2262	0.2586	0.2872	0.3210	0.3443
22	0.2212	0.2528	0.2809	0.3139	0.3367
23	0.2165	0.2475	0.2749	0.3073	0.3295
24	0.2120	0.2424	0.2693	0.3010	0.3229
25	0.2079	0.2377	0.2640	0.2952	0.3166
26	0.2040	0.2332	0.2591	0.2896	0.3106
27	0.2003	0.2290	0.2544	0.2844	0.3050
28	0.1968	0.2250	0.2499	0.2794	0.2997
29	0.1935	0.2212	0.2457	0.2747	0.2947
30	0.1903	0.2176	0.2417	0.2702	0.2899
> 40	$1.07/\sqrt{n}$	$1.22/\sqrt{n}$	$1.36/\sqrt{n}$	$1.52/\sqrt{n}$	$1.63/\sqrt{n}$

Tabelle A.1: Kritische Werte des KS-Tests aus [31]

B

BEILIEGENDE DATEIEN

Ein Datenträger mit der folgenden Ordnerstruktur ist der Arbeit beigelegt:



- In dem Ordner **application** ist der Quellcode dieser Arbeit zu finden. Dies umfasst die Schema-Evolutions-Anwendung (Master-Join), die Apache Hive-UDF für den Datenbank-Hook, die Apache Hive-UDF für das Berechnen der Editierdistanz und den Testdatengenerator.
- Eine lokale Hadoop-Platform inklusive Apache Hive kann in dem Ordner **environment** durch mehrere Docker-Images erzeugt werden. Über ein *docker-compose up* werden jene Docker-Images in einem neuen Cluster gestartet.
- Alle Dateien dieser Thesis sind in dem Ordner **thesis** zu finden. Dies umfasst neben dem LaTeX-Quellcode auch die vollständige Bibliografie inklusive nicht zitierter Literatur.
- Die Ergebnisse der Evaluation sind in dem Ordner **tests** zu finden. Neben den konsolidierten Ergebnissen in dem Ordner **summary** können die Rohdaten in dem Ordner **testcases** nachvollzogen werden. Die Ergebnisse der Laufzeituntersuchungen sind konsolidiert verfügbar.
- Das vollständige *Mission-Player*-Datenmodell und der Testdatengenerator des TPC-H-Datenmodells sind in dem Ordner **data** zu finden. In der *readme.md* wird beschrieben, wie das TPC-H-Datenmodell nach Apache Hive geladen werden kann.
- In dem Ordner **dump** sind die Rohdaten nach dem Tag der Evaluation geordnet gespeichert.

C

VERWENDETE SOFTWARE

Auch wenn ein konkreter Softwarestack für diese Arbeit unerheblich ist, sind im Folgenden die verwendeten Softwarelösungen mit deren Versionen gelistet:

Softwarename	**Version**	**Verwendung**
Kotlin	1.4.10	Programmiersprache für den Schema-Evolutions-Prozess
Spring Framework	2.3.5	Enterprise Laufzeitumgebung
Apache Hadoop	3.2.1	Skalierbare und verteilte Laufzeitumgebung für Big-Data-Anwendungen
Apache Hive	3.1.2	Skalierbare und verteilte NoSQL-Datenbank
Docker	2.5.0.1	Containertechnologie als Laufzeitumgebung
Maven	3.6.3	Softwarebuild

Tabelle C.1: Verwendeter Softwarestack

D

INSTALLATION UND DEPLOYMENT

Der Softwarestack kann mithilfe von Docker-Images aufgebaut werden. Hierfür sind die notwendigen Containerbeschreibungen in dem Ordner **environment** zu finden. Das beigelegte *make-file* übernimmt für alle Docker-Images den Build-Prozess. Dieser Prozess wird mit *make build* gestartet und mit *docker-compose* im selbigen Ordner ausgeführt. Die Schema-Evolutions-Anwendung kann mit dem Build-Management-Tool *Apache Maven* erzeugt werden:

Listing D.1: Build-Prozess

```
# Build
mvn clean package

# Start der Anwendung
docker build -t master/schemajoin SchemaJoin
```

Alternativ lässt sich das erzeugte Artefakt auch außerhalb eines Docker-Images starten:

Listing D.2: Deployment

```
  java -jar \
  -Djavax.net.ssl.trustStore="/etc/pki/ca-trust/extracted/java/cacerts" \
  -Djavax.net.ssl.trustStorePassword= \
  -Djavax.security.auth.useSubjectCredsOnly=true \
  -Djava.security.krb5.realm="" \
  -Djava.security.krb5.kdc="" \
  -Djava.security.krb5.conf="/etc/krb5.conf" \
  /var/iophome/ta2dlc1/cb2lus2/SchemaJoin-0.1-SNAPSHOT.jar \
  --spring.config.location=file:///application-coba.yml
```

Durch die Definition einer Anwendungskonfiguration kann flexibel bestimmt werden, welchen Ressourcen zu verwenden sind. Die folgende Anwendungskonfiguration ist für die Verbindung zur lokalen Laufzeitumgebung konfiguriert:

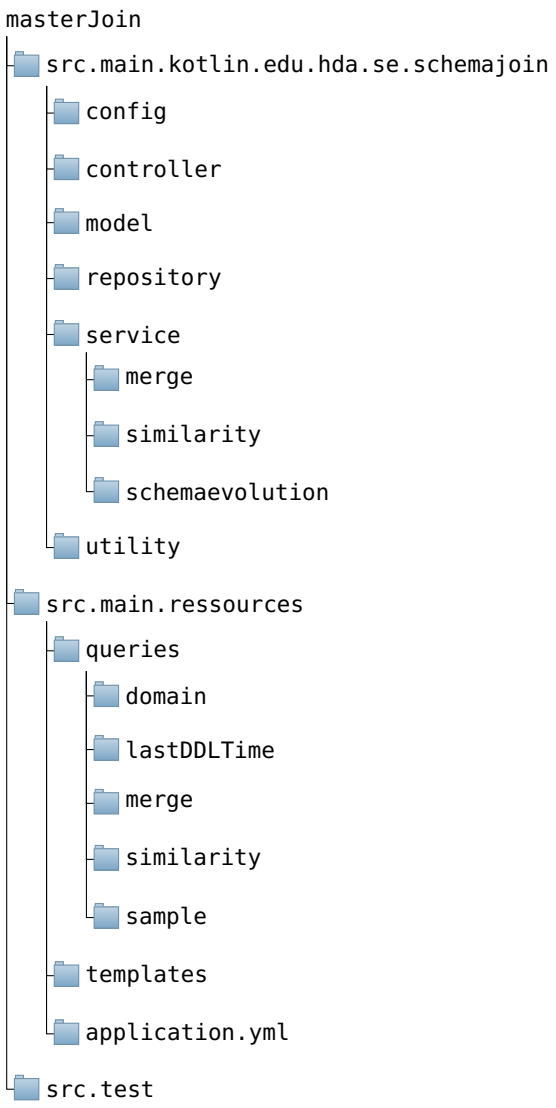
Listing D.3: Anwendungskonfiguration

```
spring:
  main.banner-mode: 'Console'
  liquibase.enabled: false

  jpa:
    hibernate:
      ddl-auto: update
      show_sql: false
      format_sql: false
      dialect: org.hibernate.dialect.PostgreSQLDialect
      naming:
        physical-strategy: org.hibernate.boot.model.naming.
            PhysicalNamingStrategyStandardImpl
        implicit-strategy: org.hibernate.boot.model.naming.
            ImplicitNamingStrategyLegacyJpaImpl
    properties.hibernate.enable_lazy_load_no_trans: true

  metastore:
    jdbc-url: jdbc:mariadb://localhost:3306/metastore
    driverClassName: org.mariadb.jdbc.Driver
    username: root
    password: root

  se:
    jdbc-url: jdbc:postgresql://localhost:5432/schemaevolution_dev
    driverClassName: org.postgresql.Driver
    username: schemaevolution
    password: schemaevolution

  thymeleaf:
    check-template-location: true
    cache: false

mqtt:
  server: tcp://localhost:1883
  topic: dev/hive
  content: hook
  isCleanSession: true
  isAutomaticReconnect: true
  keepAliveInterval: 15
  connectionTimeout: 60
  qos: 0
  response.topic: dev/hive/response

hive:
  jdbc-url: jdbc:hive2://localhost:10000
  driverClassName: org.apache.hive.jdbc.HiveDriver
  username: hive
  password: hive

server.port: 2121
mode: local
```

E

WARTUNG UND WEITERENTWICKLUNG

Über ein Webinterface kann der Status der Anwendung eingesehen werden. Das Webinterface kann über einen beliebigen Webbrowser unter der Adresse `http://127.0.0.1:2121` erreicht werden. Nach der Authentifizierung mit den Login-Daten *admin:admin* können alle Informationen für den Betrieb und die Administration der Anwendung eingesehen werden. Eine Weiterentwicklung der Verfahren dieser Arbeit ist durch eine Modifikation der SQL-Abfragen im **Ressource**-Ordner möglich. Im Ordner **schemajoin** sind das interne Datenmodell, die API und die Services (merge, similarity und schemaevolution) zu finden. Die Ordnerstruktur ist folgende:



F

ROHDATENZUGRIFF

Im Kontext der Evaluation des Commerzbank-Datenmodells sind alle Erkenntnisse in einer H2-Datenbank gespeichert worden. Diese kann mithilfe von https://www.h2database.com/html/main.html geöffnet werden. Die Login-Daten sind:

Listing F.1: Auslesen einer H2-Datenbank

```
jdbc-url: jdbc:h2:file:/var/iophome/ta2dlc1/cb2lus2/testdb
driverClassName: org.h2.Driver
username: sa
password:
```

[1] Benjamin Auer und Franz Seitz. "Funktionsbegriff und Funktionseigenschaften". In: *Grundkurs Wirtschaftsmathematik: Prüfungsrelevantes Wissen – Praxisnahe Aufgaben – Komplette Lösungswege*. ISBN: 978-3834985163. Wiesbaden: Gabler, 2010, S. 105–128. DOI: 10.1007/978-3-8349-8516-3_8.

[2] Hannes Awolin. "Verfahren zur Ermittlung von Integritätsbedingungen in NoSQL-Datenbanken". Masterarb. Universität Rostock, 2017.

[3] Mohamed Amine Baazizi, Clément Berti, Dario Colazzo, Giorgio Ghelli und Carlo Sartiani. "Human-in-the-Loop Schema Inference for Massive JSON Datasets". In: *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020*. OpenProceedings.org, 2020, S. 635–638. DOI: 10.5441/002/edbt.2020.82.

[4] Wessel Badenhorst. *Publish–Subscribe Pattern*. ISBN: 978-1484226803. Berkeley, CA: Apress, 2017. DOI: 10.1007/978-1-4842-2680-3_20.

[5] Alexander Bilke und Felix Naumann. "Schema matching using duplicates". In: *Proceedings - International Conference on Data Engineering* (2005). ISBN: 978-0769522858, S. 69–80. ISSN: 10844627. DOI: 10.1109/ICDE.2005.126.

[6] Edward Capriolo, Dean Wampler und Jason Rutherglen. *Programming Hive*. 1st. ISBN: 978-1449319335. O'Reilly Media, Inc., 2012.

[7] Hinrich Schütze Christopher D. Manning, Prabhakar Raghavan. *Introduction to Modern Information Retrieval*. ISBN: 978-0521865715. Cambridge University Press, 2008.

[8] Cecil Eng H. Chua, Roger H. L. Chiang und Ee-Peng Lim. "Instance-based attribute identification in database integration". In: *The VLDB Journal The International Journal on Very Large Data Bases* 12.3 (2003), S. 228–243. ISSN: 1066-8888. DOI: 10.1007/s00778-003-0088-y. URL: http://link.springer.com/10.1007/s00778-003-0088-y.

[9] Charles Darwin und John Murray. *On the origin of species by means of natural selection, or, The preservation of favoured races in the struggle for life*. John Murray, Albemarle Street, 1859, S. 564. URL: https://www.biodiversitylibrary.org/item/135954.

[10] Apache Software Foundation. *Apache Hive Design*. Version 3.1.2. https://cwiki.apache.org/confluence/display/Hive/Design, last accessed on 07/12/20. 2020.

[11] Thomas Frisendal. "Opportunity: NoSQL and Big Data". In: *Design Thinking Business Analysis: Business Concept Mapping Applied*. ISBN: 978-3642328442. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, S. 111–112. DOI: 10.1007/978-3-642-32844-2_15.

[12] "Kullback-Leibler Divergence". In: *Encyclopedia of Operations Research and Management Science*. Hrsg. von Saul I. Gass und Michael C. Fu. ISBN: 978-1441911537. Boston, MA: Springer US, 2013, S. 844–844. DOI: 10.1007/978-1-4419-1153-7_200372.

[13] Saurabh Gupta und Venkata Giri. *Introduction to Enterprise Data Lakes*. ISBN: 978-1484235225. Berkeley, CA: Apress, 2018. DOI: 10.1007/978-1-4842-3522-5_1.

[14] *Hackolade*. https://hackolade.com/help/WelcometoHackolade.html, last accessed on 07/12/20.

[15] Trevor Hastie, Robert Tibshirani und Jerome Friedman. "Linear Methods for Regression". In: *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. ISBN: 978-0387848587. New York, NY: Springer New York, 2009, S. 43–99. DOI: 10.1007/978-0-387-84858-7_3.

[16] Gaston C. Hillar. *MQTT Essentials - A Lightweight IoT Protocol*. ISBN: 978-1787285149. Packt Publishing, 2017.

[17] Lan Jiang und Felix Naumann. "Holistic primary key and foreign key detection". In: *Journal of Intelligent Information Systems* 54.3 (2020), S. 439–461. ISSN: 0925-9902. DOI: 10.1007/s10844-019-00562-z. URL: http://link.springer.com/10.1007/s10844-019-00562-z.

[18] Meike Klettke, Hannes Awolin, Uta Störl, Daniel Muller und Stefanie Scherzinger. "Uncovering the evolution history of data lakes". In: *2017 IEEE International Conference on Big Data*. Bd. 2018. ISBN: 978-1538627150. IEEE, 2017, S. 2462–2471. DOI: 10.1109/BigData.2017.8258204.

[19] Meike Klettke, Uta Störl, Manuel Shenavai und Stefanie Scherzinger. "NoSQL schema evolution and big data migration at scale". In: *2016 IEEE International Conference on Big Data*. ISBN: 978-1467390057. IEEE, 2016, S. 2764–2774. DOI: 10.1109/BigData.2016.7840924.

[20] Henning Köhler, Xiaofang Zhou, Shazia Sadiq, Yanfeng Shu und Kerry Taylor. "Sampling dirty data for matching attributes". In: *Proceedings of the 2010 international conference on Management of data - SIGMOD '10*. Bd. SIGMOD 10. ISBN: 978-1450300322. New York, USA: ACM Press, 2010, S. 63. DOI: 10.1145/1807167.1807177.

[21] Haridimos Kondylakis, Antonis Fountouris, Apostolos Planas, Georgia Troullinou und Dimitris Plexousakis. "Enabling Joins over Cassandra NoSQL Databases". In: *Big Data Innovations and Applications*. Hrsg. von Muhammad Younas, Irfan Awan und Salima Benbernou. ISBN: 978-3030273552. Cham: Springer International Publishing, 2019, S. 3–17.

[22] Cheng-Hung Lin, Chih-Tsun Huang, Chang-Ping Jiang und Shih-Chieh Chang. "Optimization of Pattern Matching Circuits for Regular Expression on FPGA". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 15.12 (2007), S. 1303–1310.

[23] Andreas Meier und Michael Kaufmann. "NoSQL Databases". In: *SQL & NoSQL Databases: Models, Languages, Consistency Options and Architectures for Big Data Management*. ISBN: 978-3658245498. Wiesbaden: Springer Fachmedien Wiesbaden, 2019, S. 201–218. DOI: 10.1007/978-3-658-24549-8_7.

[24] Daniel Müller. "Schema-Extraktion zur Unterstützung des Schema-Managements in NoSQL-Datenbanksystemen". Masterarb. Hochschule Darmstadt, 2016.

[25] Sameer Munir, Faraz Khan und Muhammad Atif Riaz. "An instance based schema matching between opaque database schemas". In: *2014 4th International Conference on Engineering Technology and Technopreneurship (ICE2T)*. ISBN: 978-1479946211. IEEE, 2014, S. 177–182. DOI: 10.1109/ICE2T.2014.7006242. URL: http://ieeexplore.ieee.org/document/7006242/.

[26] Thorsten Papenbrock und Felix Naumann. "A Hybrid Approach to Functional Dependency Discovery". In: *Proceedings of the 2016 International Conference on Management of Data - SIGMOD '16*. Bd. 26-June-20. 1. ISBN: 978-1450335317. New York, USA: ACM Press, 2016, S. 821–833. DOI: 10.1145/2882903.2915203. URL: http://dl.acm.org/citation.cfm?doid=2882903.2915203.

[27] Dieter Pawelczak. *Start in die Technische Informatik*. ISBN: 978-3832275402. Shaker, 2008.

[28] Erhard Rahm und Philip A. Bernstein. "A survey of approaches to automatic schema matching". In: *The VLDB Journal* 10.4 (2001), S. 334–350. ISSN: 10668888. DOI: 10.1007/s007780100057.

[29] Aeneas Rooch. "Grundlagen der Statistik". In: *Statistik für Ingenieure: Wahrscheinlichkeitsrechnung und Datenauswertung endlich verständlich*. ISBN: 978-3642548574. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, S. 71–155. DOI: 10.1007/978-3-642-54857-4_2.

[30] Gunter Saake, Andreas Heuer und Kai-Uwe Sattler. *Datenbanken: Implementierungstechniken*. 2. Aufl. ISBN: 978-3826614380. mitp, 2005.

[31] *Statistical Tables*. https://www.york.ac.uk/depts/maths/tables/kolmogorovsmirnov.htm, last accessed on 07/12/20. 2020.

[32] Thomas Studer. "Die relationale Algebra". In: *Relationale Datenbanken: Von den theoretischen Grundlagen zu Anwendungen mit PostgreSQL*. ISBN: 978-3662465714. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, S. 41–68. DOI: 10.1007/978-3-662-46571-4_4.

[33] TPC. "TPC-H Benchmark". In: *Transaction Processing Performance Council TPC* (2011). http://www.tpc.org/tpch/spec/tpch2.8.0.pdf, last accessed on 07/12/20, S. 1–134.

[34] Andreas François Vermeulen. "Data Science Technology Stack". In: *Practical Data Science: A Guide to Building the Technology Stack for Turning Data Lakes into Business Assets*. ISBN: 978-1484230541. Berkeley, CA: Apress, 2018, S. 1–13. DOI: 10.1007/978-1-4842-3054-1_1.

[35] Verordnung (EU) 2016/679. *Verordnung (EU) 2016/679 des Europäischen Parlaments und des Rates vom 27. April 2016 zum Schutz natürlicher Personen bei der Verarbeitung personenbezogener Daten, zum freien Datenverkehr und zur Aufhebung der Richtlinie 95/46/EG (Datenschutz-Grundverordnung)*. 2016.

[36] Deepak Vohra. *Practical Hadoop Ecosystem: A Definitive Guide to Hadoop-Related Frameworks and Tools*. 1st. ISBN: 978-1484221982. Berkely, CA, USA: Apress, 2016. DOI: 10.1007/978-1-4842-2199-0. URL: https://link.springer.com/book/10.1007%2F978-1-4842-2199-0.

[37] Martin H. Weik. "entity type". In: *Computer Science and Communications Dictionary*. ISBN: 978-1402006135. Boston, MA: Springer US, 2001, S. 526–526. DOI: 10.1007/1-4020-0613-6_6288.

[38] Hong Yao und Howard J. Hamilton. "Mining functional dependencies from data". In: *Data Mining and Knowledge Discovery* 16.2 (2008), S. 197–219. ISSN: 1384-5810. DOI: 10.1007/s10618-007-0083-9. URL: http://link.springer.com/10.1007/s10618-007-0083-9.

[39] Young-Gook Ra und E.A. Rundensteiner. "A transparent schema-evolution system based on object-oriented view technology". In: *IEEE Transactions on Knowledge and Data Engineering* 9.4 (1997), S. 600–624. ISSN: 10414347. DOI: 10.1109/69.617053. URL: http://ieeexplore.ieee.org/document/617053/.

[40] Liang Zhao, Sherif Sakr, Anna Liu und Athman Bouguettaya. *Introduction*. ISBN: 978-3319047652. Cham: Springer International Publishing, 2014, S. 1–7. DOI: 10.1007/978-3-319-04765-2_1.