Universität Hamburg, Fakultät Wirtschafts- und Sozialwissenschaften

Dissertation zur Erlangung des Grades eines Doktors
der Wirtschafts- und Sozialwissenschaften der Universität Hamburg,
Fakultät Wirtschafts- und Sozialwissenschaften
(Rechtsnachfolge der HWP-Hamburger Universität für Wirtschaft und Politik)

Implementierung eines Datenqualitätsdienstes zur evolutionären Datenqualitätsverbesserung in relationalen Datenbankmanagementsystemen

vorgelegt von Sönke Cordts

am 03. März 2008

bei Prof. Dr. Gerhard Brosius (1. Gutachter)
und Prof. Dr. Manfred Sommer (2. Gutachter)

wissenschaftliches Gespräch am 27. Februar 2009

Zusammenfassung

Seit Mitte der 90er Jahre werden zunehmend Anwendungssysteme entwickelt, die Daten gemeinsam nutzen oder unterschiedliche Datenbestände zu einem gemeinsamen Datenbestand migrieren. Werden Daten von mehreren Anwendungssystemen genutzt oder Datenbestände integriert, so ist die Qualität der Daten von entscheidender Bedeutung. Mangelnde Datenqualität war in der Vergangenheit allerdings ein wesentlicher Grund dafür, dass solche Softwareprojekte gescheitert sind.

Die vorliegende Arbeit thematisiert das Problem der Datenqualität und beschreibt Verfahren aus unterschiedlichen Bereichen der Informatik (u.a. Datenbanken, Information Research, Bioinformatik, Künstliche Intelligenz), die zur Analyse und zum Verbessern von Daten wichtig sind. Darauf aufbauend wird eine Architektur vorgeschlagen, wie Datenqualitätsdienste in die Architektur eines relationalen Datenbankmanagementsystem (RDBMS) integriert werden können. Über die Komponente der Regelverwaltung kann die Datenqualität gerade im Hinblick auf Legacy-Anwendungen über ein evolutionäres Vorgehen verbessert werden, indem Geschäftsregeln in der Datenbank gespeichert werden, ohne dass Anwendungen geändert werden müssen. Da Datenqualität domänenspezifische Verfahren benötigt, liegt ein weiterer Schwerpunkt der Architektur in der flexiblen Erweiterung der Dienste.

Schließlich werden die Umsetzung dieser Architektur in ein bestehendes relationales Datenbankmanagementsystem anhand eines Prototypen aufgezeigt und verschiedene Szenarien zur Verbesserung der Daten erläutert.

In der Arbeit wird gezeigt, dass über standardisierte SQL-Sprachkonstrukte viele Verfahren zur Verbesserung der Datenqualität integriert werden können. Gerade die Kombination der SQL mit benutzerdefinierten Funktionen bietet dabei, wie im Prototypen zu sehen, eine leistungsfähige Möglichkeit zur Verbesserung und Analyse von Daten. Vorteilhaft ist zudem die Mächtigkeit mit der das Datenqualitäts-Framework über Plug-Ins mit neuen und abgeleiteten Klassen erweitert werden kann.

Abstract

Since the mid 90s, an increasing number of application systems have been developed, which use shared datasets or different datasets as a common migrated dataset. When datasets are used by several application systems or many datasets are integrated, the quality of the data is of vital importance. In the past insufficient data quality has been an essential reason for the failure of such software projects.

The present work picks out as the problem of data quality a central theme and describes methods from different areas of informatics (among other things Databases, Information Research, Bioinformatics, Artificial Intelligence) which are suited for analysing and improving data. On this basis an architecture is proposed, how to integrate data quality services into the architecture of a relational database management system (RDBMS). With a rule management component data quality can be improved evolutionarily, especially in view of Legacy applications, by storing business rules in the database, without changing existing application systems. Because data quality needs procedures specific to domains, another main focus of the architecture lies in the flexible extension of the services.

Finally, the implementation of this architecture into an existing relational database management system with a prototype is described and different scenarios, how to improve the quality of data, are discussed.

The present work shows that with standardised SQL-Elements many procedures can be integrated to improve the data quality. Especially the combination of SQL with user-defined functions offers an efficient possibility to improve and analyse data as shown in the prototype. A further advantage lies in the possibility to expand the framework with Plug-Ins by implementing new and derived classes.

Vorwort

Entwickelt man Anwendungssysteme, so wird man immer wieder mit der Integration, Zusammenführung oder dem Austausch von Daten konfrontiert. Jeder, der das kennt, weiß, wie mühselig es ist, Datenprobleme zunächst zu erkennen und dann zu beheben. Vor allem bei der Entwicklung von Data Warehouse Systemen, bei denen unterschiedliche Datenbestände zu einem gemeinsamen Datenbestand integriert werden sollen, ist das Erkennen von Problemen ohne entsprechende Softwarewerkzeuge häufig eine wahre „Sisyphos"-Arbeit. Ist man hier nur auf herkömmliche SQL-Anweisungen angewiesen, so bleiben die Erkenntnisse über Probleme eben doch nur oberflächlich. Ein Anspruch dieser Arbeit ist es daher, Ansätze aufzuzeigen, wie man relationale Datenbankmanagementsysteme um Datenqualitätsverfahren erweitern kann.

Auf die Erstellung der Arbeit haben verschiedene Personen Einfluss gehabt. Vor allem bedanken möchte ich mich bei Herrn Prof. Dr. Gerhard Brosius und Herrn Prof. Dr. Manfred Sommer für die Betreuung der Arbeit und für die intensive Heranführung an die Thematik von Data Warehouse Systemen, sowie die Diskussionen über aktuelle Wirtschaftsinformatikthemen, bei Fau Dipl.-Betriebswirtin (FH) Maren Nasutta für die unermüdliche Durchsicht und Korrektur meiner Arbeit, bei Herrn Dr. Rüdiger Cordts für die Durchsicht und Korrektur vor allem in Hinblick auf statistische Verfahren und bei meinem langjährigen Kollegen Sebastian Richter für die zahlreichen positiven Diskussionen und auch Auseinandersetzungen über die „richtige" Art, Software zu modellieren und zu entwerfen.

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abb.	Abbildung
ADO	ActiveX Data Objects
ANSI	American National Standards Institute
AXL	Aggregate Extension Language
BI	Business Intelligence
BRE	Business Rules Engines
bzw.	beziehungsweise
CLI	Call Level Interface
CLR	Common Language Runtime
CODASYL	Conference on Data Systems Languages
CRM	Customer Relationship Management
d.h.	das heißt
DIN	Deutsches Institut für Normung
Diss.	Dissertation
DBMS	Datenbankmanagementsystem
DBS	Datenbanksystem
DBTG	Data Base Task Group
DF	Document Frequency
EAN	European Article Number
ECA	Event Condition Action
EPC	Electronic Product Code
ERP	Enterprise Ressource Planning
ETL	Extract, Transform, Load
GUAM	Generalized Update Access Method
ggf.	gegebenenfalls
HMM	Hidden Markov Modell
Hrsg.	Herausgeber
i.d.R.	in der Regel
IDF	Inverse Document Frequency
IDS	Integrated Data Store
IE	Information Extraction
IS	Informationssystem

ISBN	International Standard Book Number
ISO	International Organization for Standardization
IT	Informationstechnik
JRT	Java Routines and Types
LCS	Longest Common Subsequence
MADM	Multiple Attribute Decision Making
MED	Management of External Data
MUC	Message Understanding Conference
NAA	North American Aviation
NCD	Normalized Compression Distance
NER	Named Entity Recognition
NLP	Natural Language Programming
NYSIIS	York State Identification and Intelligence System
o.A.	ohne Angabe
o.J.	ohne Jahr
o.S.	ohne Seite
o.V.	ohne Verlag
OLAP	Online-Analytical-Processing
OLB	Object Language Binding
OMG	Object Management Group
ONCA	Oxford Name Compression Algorithm
POS	Parts-Of-Speech
PSM	Persistent Stored Modules
RDBMS	relationales Datenbankmanagementsystem
RPC	Remote Procedure Call
RW	Realwelt
S.	Seite
SAW	Simple Additive Weighting
SCM	Supply Chain Management
SEQUEL	Structured English Query Language
SGML	Standard Generalized Markup Language
SNM	Sorted-Neighborhood Methode
SOA	Service Oriented Architecture
SOX	Sarbanes-Oxley Act

SPARC	Standards Planning amd Requirements Committee
SQL	Structured Query Language
Tab.	Tabelle
TDQM	Total Data Quality Management
TF	Term Frequency
TF-IDF	Term Frequency - Inverse Document Frequency
TQdM	Total Quality data Management
TQM	Total Quality Management
u.a.	unter anderem, und andere
UDM	unternehmensweites Datenmodell
UEA	University of East-Anglia
UPC	Universal Product Code
usw.	und so weiter
vgl.	vergleiche
XML	Extensible Markup Language
z.B.	zum Beispiel
z.T.	zum Teil
z.Z.	zurzeit

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einführung

1.1 Motivation und Problemstellung

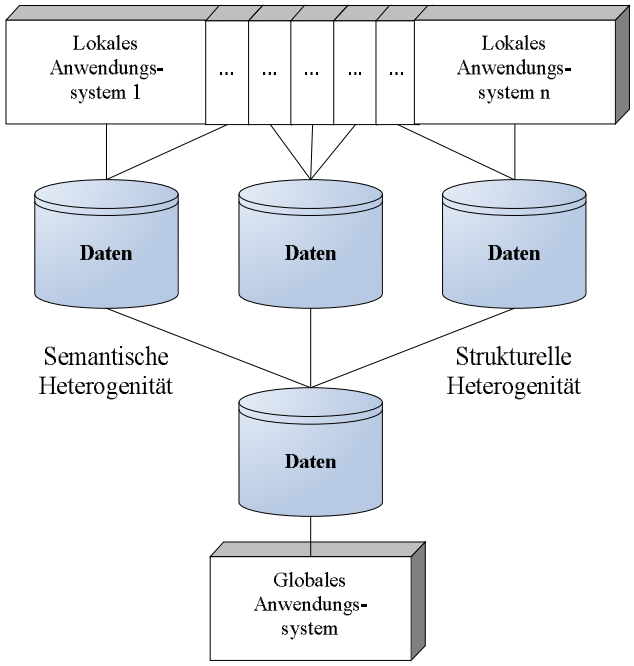
Hat man vor etwa 15 Jahren in der Anwendungsentwicklung noch in sich abgeschlossene Monolithen entwickelt, die ihre Daten direkt über Datenbanksysteme speicherten, so führten Technologieentwicklungen in den 90er Jahren zur Möglichkeit, Daten zwischen Anwendungen und Systemen auszutauschen bzw. gemeinsam zu verwenden. Dabei war die Entwicklung neuer Technologien notwendig, um die mit der Interoperabilität zwischen den Systemen einhergehenden Probleme zu beheben. Hier sind vor allem Middleware, Mediator-basierte Systeme, föderierte Datenbanksysteme, Datenintegration, ETL (Extraktion, Transformation, Laden) und aktuell dienstorientierte Architekturen (Service Oriented Architecture, SOA) als wichtige Techniken zu nennen.

Diesen neuen Techniken ist gemeinsam, dass sie Daten aus verschiedenen Datenhaltungssystemen zusammenführen und dabei die Heterogenität der Datenstruktur und der Dateninhalte in homogene Datenstrukturen und -inhalte überführen. Ein in sich geschlossenes Anwendungssystem speicherte und verwaltete seine Daten so, wie es für seine spezifische Anwendung und seine Geschäftsprozesse weitgehend optimal war. Die Daten waren dadurch häufig nicht ohne weiteres für ein integriertes Anwendungssystem geeignet. Daraus ergaben sich zwei Anforderungen an die neuen Technologien: Zum einen mussten die unterschiedlichen Datenstrukturen zusammengeführt und zum anderen Datenprobleme, z.B. Unterschiede zwischen den gespeicherten Daten und den wahren Daten, synchronisiert werden. Die Integration der Daten führte damit zu einer Fokussierung auf das eigentliche Problem der Datenqualität.

Verursachen heterogene Datenbestände vor allem Probleme in der Zusammenführung der Datenstruktur, so enthalten einzelne geschlossene Datenbestände Datenfehler wie Falscheingaben, Duplikate usw. und sind durchaus nicht homogen. M. Cochinwala unterscheidet hierbei zwischen struktureller Heterogenität – wenn Attribute unterschiedlich in verschiedenen Datenbeständen definiert sind – und semantischer Heterogenität – wenn gleich strukturierte Attribute unterschiedliche Semantik und Werte in verschiedenen Datenbeständen

verwenden.[1] Um Probleme in der Semantik der Daten zu beheben, ist ein Verständnis der Daten notwendig, das nicht automatisch ermittelt werden kann.[2] Benötigt eine globale Anwendung Daten aus unterschiedlichen Datenbeständen, die unabhängig voneinander verwaltet und entwickelt wurden, dann sind also vor allem Probleme in der Datenstruktur, in der Semantik und in der Repräsentation der Daten zu erwarten.[3]

Abb. 1: Semantische und strukturelle Heterogenität von Daten



Quelle: eigene Darstellung

Entwickelt ein Unternehmen also globale datenintegrierende Anwendungen wie ein „Data Warehouse“, ein „Customer Relationship Management“ (CRM) System oder auch nur die Anbindung ihres Warenwirtschaftssystems an eine Webanwendung für Endkunden, so werden Probleme in der Datenqualität offensichtlich, da verschiedene Alt-Anwendungen mit unterschiedlicher Semantik der Daten und einer heterogenen Struktur der Datenbasis zusammengebracht werden müssen. Durch die weltweite Vernetzung und Verbreitung seit Anfang der 90er Jahre haben deshalb gerade datenintegrierende Anwendungen einen erheblichen Entwicklungsschub erhalten. Datenintegration setzt aber immer auch eine der Anwendung angemessene Datenqualität voraus, um eine Integration zu ermöglichen. Auch

[1] Cochinwala, M.; Dalal, S.; Elmagarmid, A. K.; Verykios, V. S.: Record matching: Past, present and future; Technical Report TR-01-013; Purdue University, Department of Computer Sciences; 2001; S. 4
[2] Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 6
[3] Cochinwala, M.; Dalal, S.; Elmagarmid, A. K.; Verykios, V. S.: Record matching: Past, present and future; Technical Report TR-01-013; Purdue University, Department of Computer Sciences; 2001; S. 4

wenn Daten für eine lokale Anwendung durchaus angemessen sein können, kann eine Datenintegration Probleme in der Datenqualität offenkundig machen, die diese schlichtweg unmöglich macht. Gerade im Bereich Data Warehouse[4], CRM und SCM („Supply Chain Management") sind deshalb in den letzten Jahren viele Projekte nicht erfolgreich abgeschlossen worden.[5] Nach einer Studie der Meta Group scheitern mehr als 35% aller IT-Projekte an mangelnder Datenqualität.[6]

Letztendlich ist das Problem der Datenqualität aber vor allem auch bei in sich geschlossenen Anwendungen vorhanden. Eine Fokussierung auf das Thema Datenqualität sowohl in der Forschung als auch im kommerziellen Bereich wurde jedoch erst durch die Probleme bei der Datenintegration unterschiedlicher Datenbestände aktuell. Bis dato hat dieses Problem in den Unternehmen und in der Forschung eher nachrangige Bedeutung. In den letzten Jahren entstanden zwar Anwendungen, mit denen sich Datenbestände analysieren und Datenqualitätsprobleme bereinigen lassen[7] sowie Erweiterungen in den oben erwähnten Techniken, vor allem bei den ETL-Tools. Eine strategische Ausrichtung auf Datenqualität und die Behandlung von Daten als Unternehmensressource ist in den Unternehmen zurzeit aber nur selten und wenn dann lediglich ansatzweise vorhanden. Ein Konzept zum Datenqualitätsmanagement wurde bisher nur von etwa 20% der Unternehmen umgesetzt.[8] Das Ausmaß und die Ursachen unzureichender Datenqualität sind in den Firmen daher meistens unbekannt und unerkannt. L.P. English spricht in diesem Zusammenhang von den „normal cost of doing business".[9] Der „Report on Data Quality" des Data Warehousing Institute schätzt diese „normal costs" aufgrund von Datenqualitätsproblemen auf etwa 600 Milliarden Dollar in

[4] In einer Studie des "Business Application Research Center" (BARC) wird Datenqualität als wichtigste Eigenschaft von Business-Intelligence-Software genannt (Friedrich, D.: Einfach soll es sein - bei hoher Datenqualität; erschienen in: is report; 11. Jahrgang, 4/2007; Oxygon Verlag; München; S. 29)

[5] Beispiele hierzu finden sich in: Redman, T.: Data Quality for the Information Age; Artech House; Norwood; 1996
English, L.P.: Improving Data Warehouse and Business Information Quality; Wiley Computer Publishing; New York; 1999
Redman, T.: Data Quality - The Field Guide; Digital Press; Boston; 2001

[6] Hipp, J.: Datenqualität - Ein zumeist unterschätzer Erfolgsfaktor; erschienen in: Dadam, P.; Reichert, M. (Hrsg.): Informatik 2004 - Informatik verbindet, Band 1, Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V.; Bonner Köllen Verlag, Bonn, 2005; S. 232

[7] Vgl. hierzu: Mathes, T.; Bange, C.; Keller, P.: Software im Vergleich: Datenqualitätsmanagement, Studie des Business Application Research Center (BARC); Oxygon Verlag; München; 2005

[8] Mathes, T.; Bange, C.; Keller, P.: Software im Vergleich: Datenqualitätsmanagement; Oxygon Verlag; München; 2005; S. 1

[9] Lee, Y.W.; Pipino, L.L.; Funk, J.D.; Wang, R.Y.: Journey to Data Quality; MIT Press; Cambridge, Massachusetts; 2006; S. 14

amerikanischen Unternehmen[10] und T. Redman schätzt diese Mehrkosten der „normal costs“ durch Datenqualitätsprobleme auf 8 bis 12 Prozent der Unternehmensumsätze.[11]

Aus Sicht der Unternehmen ist das Problem der Datenqualität zwar erkannt, wird aber vorwiegend nur im Rahmen integrierender Systeme einmalig innerhalb eines abgeschlossenen Projektes behandelt. So gehen nach einer Untersuchung des SAS Institute nur 18% der befragten Unternehmen von der Korrektheit und Vollständigkeit ihrer Daten aus.[12] Nach einer Erhebung von S. Augustin werden Probleme mit der Informationsqualität bereits am vierthäufigsten genannt, die die tägliche betriebliche Arbeit am meisten behindern.[13]

Auch vom Gesetzgeber wird das Problem der Datenqualität durchaus gesehen. So wurden inzwischen Auflagen erlassen, um zu einer verbesserten Datenqualität zu kommen. In den USA gibt es seit 2002 ein Gesetz, das Richtlinien der Datenqualität für Regierungsstellen im „Data Quality Act“ festlegt.[14] Aktuelle Gesetze wie Basel II und der Sarbanes-Oxley Act (SOX) spiegeln die gesetzliche Notwendigkeit wider, sich mit der Qualität der Daten im Unternehmen auseinanderzusetzen.[15]

Datenintegrierende Geschäftspraktiken haben zwar dazu geführt, dass Unternehmen ihre Daten besser verstehen und auch verwalten,[16] ein präventives Datenqualitätsmanagement, das sich der Datenqualität bereits bei der Entstehung annimmt, existiert nicht und ist mit den bestehenden Techniken auch schwer umzusetzen.

---

[10] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 2

[11] Mathes, T.; Bange, C.; Keller, P.: Software im Vergleich: Datenqualitätsmanagement; Oxygon Verlag; München; 2005; S. 15

[12] Bange, C.: Das neue Gesicht der Datenintegration; Business Application Reseach Center (BARC); http://www.competence-site.de/datenbanken.nsf/9D457579637252C7C1256E6F0033EFDC/$File/datenintegration_barc_0304.pdf; 2004; S. 6

[13] Augustin, S.: Der Stellenwert des Wissensmanagement im Unternehmen; erschienen in: Mandl, H.; Reinmann-Rothmeier, G. (Hrsg.): Wissensmanagement: Informationszuwachs - Wissensschwund? Die strategische Bedeutung des Wissensmanagements; München; Oldenbourg; 2000; S. 162

[14] Dippold, R.; Meier, A.; Schnider, W.; Schwinn, K.: Unternehmensweites Datenmanagement, 4. Auflage; Vieweg; Braunschweig, Wiesbaden; 2005; S. 249

[15] Vgl. hierzu: Behme, W.; Nietzschmann, S.: Strategien zur Verbesserung der Datenqualität im DWH-Umfeld; erschienen in: HMD - Praxis der Wirtschaftsinformatik; dpunkt Verlag; Heidelberg; S. 43
Lüssem, J.; Tasev, P.; Tiedemann-Muhlack, M.: Ein lernendes System zur Verbesserung der Datenqualität und Datenqualitätsmessung; erschienen in: Cremers, A. u.a. (Hrsg.): Beiträge der 35. Jahrestagung der Gesellschaft für Informatik e.V., Informatik 2005 - Informatik LIVE!, Band 2, Bonner Köllen Verlag, Bonn, 2005; S. 418

[16] Lee, Y.W.; Pipino, L.L.; Funk, J.D.; Wang, R.Y.: Journey to Data Quality; MIT Press; Cambridge, Massachusetts; 2006; S. 203

Wissen über die Unternehmensdaten ist notwendig, um Datenqualitätsprobleme zu erkennen, zu beheben und letztlich zu vermeiden. Da Datenqualitätsmethoden häufig in Zusammenhang mit datenintegrierenden Themen, vor allem Data Warehouse, Data Mining und Datenintegration untersucht werden, liegt auch der Schwerpunkt der Forschung in diesen Anwendungsbereichen, aber nicht in der Vermeidung von Datenqualitätsproblemen bei der Entstehung. Datenqualität kann in einem integrierten Anwendungssystem immer nur so gut sein, wie der Geschäftsprozess, der die Daten erzeugt. Nach L. P. English ist es deshalb entscheidend, die Geschäftsprozesse zu beobachten und zu verändern, die Datenqualitäts-mängel verursachen. Neben organisatorischen Veränderungen gehören hierzu vor allem auch technische Änderungen wie Echtzeitvalidierung usw.[17]

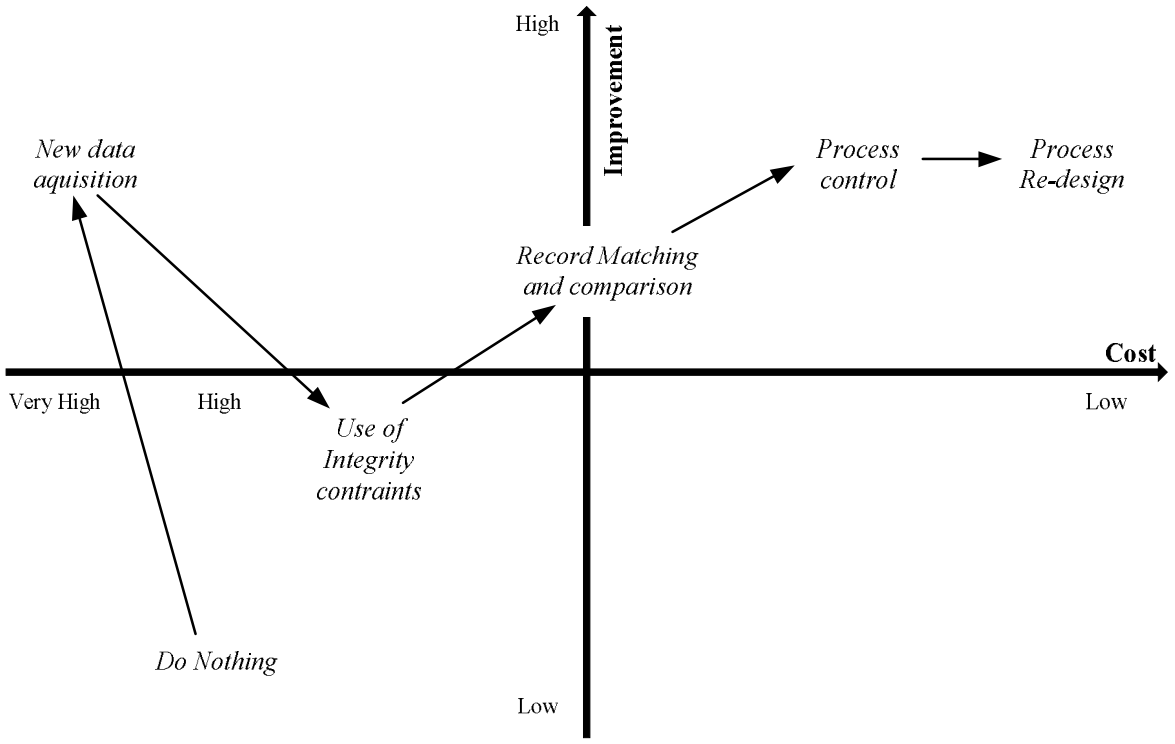
C. Batini und M. Scannapieco beschreiben in diesem Zusammenhang sechs Strategien zur Datenqualitätsverbesserung[18], vom „Nichtstun" („Do nothing") bis zum „Prozess Re-Design". Beim „Nichtstun" verschlechtert sich sowohl die Qualität der Daten und damit auch die Qualität des Geschäftsprozesses. Eine erste Strategie liegt in der Neu-Anschaffung der Daten und damit in deren Bereinigung („New Data Acquisition"). Dieser Vorgang muss periodisch wiederholt werden, um die Qualität der Daten zu erhalten, womit hohe Kosten verbunden sind. Ein nächster Schritt ist die Verwendung von Integritätsbedingungen, wodurch sich die Kosten verringern. Allerdings können nur solche Fehler verhindert werden, die über Integritätsbedingungen überprüft werden können. Als nächste verbesserte Strategie steht das Record Matching, also das Bereinigen einer Datenbasis von Duplikaten. Erst eine prozessgetriebene Methode schließlich führt nach C. Batini und M. Scannapieco zu effektiven Verbesserungen in der Datenqualität. Hierzu gehört zunächst die Prozess-Steuerung (Process control) und darauf aufbauend der Prozess Re-Design.[19]

[17] Vgl. hierzu: English, L.P.: Improving Data Warehouse and Business Information Quality; Wiley Computer Publishing; New York; S. 286f

[18] Einen kompakteren Ansatz beschreibt T. Redman, indem er zwischen dem „clean-up" Ansatz und dem Prozessansatz unterscheidet. Im ersten Schritt werden die Daten und im zweiten Schritt die Prozesse verbessert, die diese Daten erzeugen (Vgl. hierzu: Redman, T.: Data Quality for the Information Age; Artech House; Norwood; 1996; S. xxiv)

[19] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 166f

Abb. 2: Strategien zur Verbesserung der Datenqualität



Quelle: Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 166

Das einmalige Überprüfen der Daten („New data aquisition“ und „Record Matching and comparison“) ist zwar notwendig, aber allein natürlich nicht ausreichend. Ein einmaliger Bereinigungsprozess kann Symptome nur kurzfristig beheben, aber Datenprobleme nicht langfristig beseitigen. Unternehmen müssen proaktiv Lösungen entwickeln, um Datenprobleme zu verhindern. Dazu benötigen Sie Wissen über den Datenproduktionsprozess, warum dieser eventuell fehlschlägt bzw. unzulängliche Ergebnisse liefert. Hierzu müssen prozessorientierte Techniken im Datenproduktionsprozess angewendet werden.[20] Nach einer empirischen Erhebung von W. Eckerson wurden mit 76% vor allem die Dateneingaben von Mitarbeitern als Ursache für Datenqualitätsprobleme genannt.[21] Die Probleme liegen also hauptsächlich bei der Entstehung der Daten in den Erfassungsprozessen. Eine Lösung hierfür kann sich nur durch Betrachtung des Gesamtprozesses von der Datenentstehung bis zur

[20] Vgl. hierzu: Lee, Y.W.; Pipino, L.L.; Funk, J.D.; Wang, R.Y.: Journey to Data Quality; MIT Press; Cambridge, Massachusetts; 2006; S. 106ff

[21] Mathes, T.; Bange, C.; Keller, P.: Software im Vergleich: Datenqualitätsmanagement; Oxygon Verlag; München; 2005; S. 10

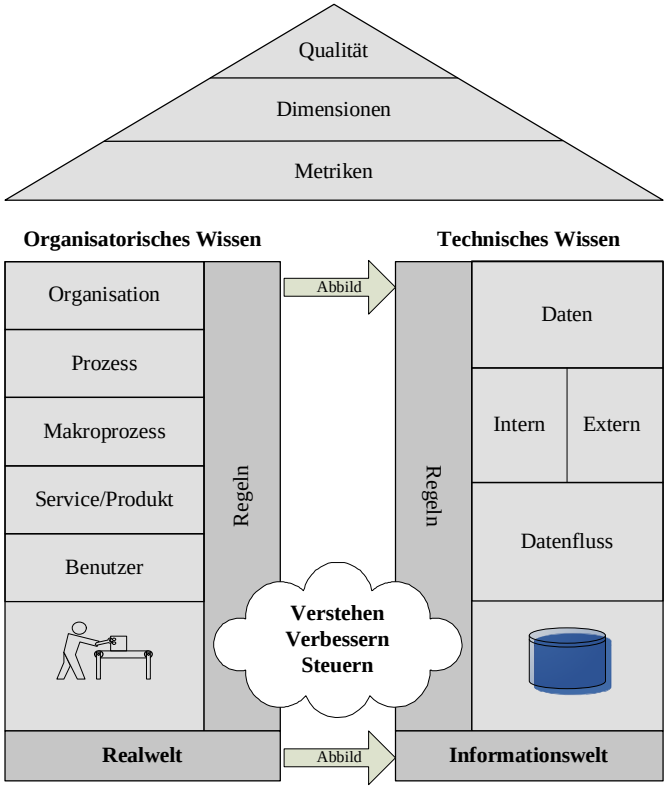
Datenverwendung finden,[22] indem man die Ursachen der Datenqualitätsmängel identifiziert und Maßnahmen zur Behebung umsetzt. Langfristig entstehen so auch weniger Kosten, da ansonsten Datenbereinigungsmaßnahmen kontinuierlich in bestimmten Zeitabständen durchgeführt werden müssten. Eine langfristig hohe Datenqualität kann aber nur durch eine Echtzeitvalidierung der Daten bereits während der Erfassung erfolgen.[23] Das setzt wiederum eine hohe Qualität der Software voraus, um Datenmängel zu vermeiden. Datenqualität wird also nicht nur durch eine unangemessene Behandlung der Daten (z.B. fehlende oder falsche Daten, die nicht der Realwelt entsprechen, Dubletten, logische Fehler) beeinflusst, sondern als zweiter wesentlicher Faktor natürlich auch durch Probleme im Design des Datenbestandes und damit in der darauf aufsetzenden Software. Zu Problemen im Design gehören z.B. fehlende Normalisierung bzw. eine bewusste Denormalisierung einer Datenbank oder fehlende Integritätsbedingungen. Ist ein Softwaresystem unzureichend entwickelt worden, d.h. wurden Geschäftsregeln, Integritätsbedingungen usw. ignoriert und ist damit der Ausschnitt der realen Welt, der sich im Softwaresystem widerspiegeln soll, ungenügend abgebildet, so führt dies zwangsläufig zu einer unzureichenden Datenqualität. Nach einer Untersuchung der Standish Group werden in den USA jährlich 250 Milliarden Dollar für die Entwicklung von Softwaresystemen ausgegeben. Dabei werden nur 16% der Projekte termin- und budgetgerecht beendet, 31% werden vorwiegend wegen qualitativer Mängel abgebrochen und 53% überschreiten die Kosten um 189%. Die dann fertig gestellten Systeme erfüllen nur etwa 42% der an sie gestellten Anforderungen.[24]

Die Zahlen zeigen, dass Datenqualität nur über eine entsprechende Softwarequalität erreicht werden kann. Die Entwicklung einer Software besteht – einfach ausgedrückt – in der Abbildung eines Ausschnittes der Realwelt (Organisatorisches Wissen mit Prozessen, Informationen, Geschäftsregeln) in die Informationswelt (Technisches Wissen mit Daten, Datenfluss, Geschäftsregeln). Je größer dabei die Unterschiede zwischen der Realwelt und der Informationswelt, umso gravierender sind normalerweise die Mängel an der Software und damit an den Daten. Diese Unterschiede zu erkennen und zu verstehen ist Voraussetzung, um die entsprechenden Geschäftsprozesse und deren Abbildung in der Software zu verbessern und schließlich zu steuern.

[22] Helfert, M.: Proaktives Datenqualitätsmanagement in Data-Warehouse-Systemen, Dissertation; logos-Verlag; Berlin; 2002; S. 4
[23] Mathes, T.; Bange, C.; Keller, P.: Software im Vergleich: Datenqualitätsmanagement; Oxygon Verlag; München; 2005; S. 21
[24] Greenfield, J.; Short, K.: Software Factories - Moderne Software-Architekturen mit SOA, MDA, Patterns und agilen Methoden; Redline; Heidelberg; 2006; S. 25

.

Abb. 3: Abbildung der Realwelt in die Informationswelt



Quelle: eigene Darstellung in Anlehnung an: Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 162

Für neu zu entwickelnde Anwendungssoftware auf der „grünen Wiese“ wäre es zwar möglich, Klassenbibliotheken oder Frameworks, die Datenqualitätsaspekte berücksichtigen, zu verwenden. Vor allem Probleme bei Alt-Anwendungen würde man damit aber nicht lösen und diese stellen das größere Problem dar. Ein sinnvoller Ansatz wäre die Integration von Datenqualitätsaspekten in bestehende Datenbankmanagementsysteme (DBMS), da diese von den meisten Anwendungssystemen zur Speicherung ihrer Daten verwendet werden. Das relationale Datenmodell ist heutzutage das primäre Datenmodell in datenverarbeitenden Anwendungen und hat sich vor allem wegen seiner Einfachheit im Vergleich zu vorherigen Modellen (hierarchisch, Netzwerkmodell) durchsetzen können.[25] Etwa 75% der heute in Unternehmen eingesetzten Datenbankmanagementsysteme sind relationale Datenbank-managementsysteme (RDBMS)[26]. Zwar bieten kommerzielle RDBMS-Produkte zusätzliche

[25] Silberschatz, A.; Korth, H.F.; Sudarshan, S.: Database System Concepts, Fifth Edition; McGraw-Hill; New York; 2006; S. 37
[26] Vgl. hierzu: Cordts, S.: Datenbankkonzepte in der Praxis; Addison-Wesley; München; 2002; S. 11

Software (z.B. ETL-Tools) an, die Komponenten zur Verbesserung der Datenqualität enthalten. In der Regel sind diese jedoch eher für Einmalprojekte im Bereich der Datenintegration entwickelt worden und bieten keine dauerhafte Verbesserung der Datenqualität für bestehende Systeme. Im kommerziellen Bereich existieren daneben spezielle Datenqualitäts-Tools, die hauptsächlich relationale Datenbankmanagementsysteme unterstützen und auch eher auf Einmalprojekt wie vor allem „Data Warehouses" ausgerichtet sind.

Relationale Datenbankmanagementsysteme unterstützen bereits zahlreiche Integritätsbedingungen im SQL-Standard. Hier stellt sich die Frage, warum Datenqualitätsprobleme trotzdem in einem so großen Ausmaß existieren. Ein Grund dafür ist sicherlich, dass in der Praxis die Rolle und die Bedeutung von Integritätsbedingungen häufig unterschätzt wird,[27] da man normalerweise davon ausgeht, dass diese im Anwendungssystem realisiert werden. Das führt dazu, dass sich selbst einfache Integritätsbedingungen nicht im Datenbankdesign wiederfinden, sondern im Anwendungssystem implementiert werden. Ein weiterer Grund könnte darin zu finden sein, dass die Integritätsbedingungen für eine Datenqualitätsverbesserung nicht ausreichen, da z.B. eine Duplikaterkennung nicht unmittelbar in SQL berücksichtigt werden kann.

Zwar gab es in den letzten Jahren viele neue Entwicklungen in der Datenbanktechnologie, um Sachverhalte der realen Welt genauer und umfassender in einer Datenbank abzubilden (hier sind vor allem die objektorientierten Datenbanken und die objektrelationalen Erweiterungen der relationalen Datenbankmanagementsysteme zu nennen),[28] doch eine Verbesserung der Datenqualität hat sich dadurch bisher nicht ergeben. Das liegt u.a. darin begründet, dass im wissenschaftlichen Bereich Datenqualität eher aus organisatorischer Sicht betrachtet wurde[29] und im technisch-konzeptionellen Bereich eher Spezialthemen wie Datenintegration und Duplikatbereinigung erforscht wurden.

[27] Türker, C.; Gertz, M.: Semantic integrity support in SQL:1999 and commercial (object-)relational database management systems; erschienen in: The VLDB Journal - The international Journal on Very Large Data Bases, Vol. 10, Issue 4; Springer Verlag; New York; 2001; S. 266

[28] Dittrich, K.R.; Gatziu, S.: Aktive Datenbanksysteme - Konzepte und Mechanismen, 2. Auflage; dpunkt Verlag; Heidelberg; 2000; S. 1

[29] Vgl. hierzu: English, L.P.: Improving Data Warehouse and Business Information Quality: Wiley Computer Publishing; New York; 1999

1.2 Ziel der Arbeit

Betrachtet man bisherige Ansätze, so sind diese stark von der Datenintegration geprägt. Lösungen zur Vermeidung von Datenqualitätsproblemen bereits bei der Entstehung werden in der Regel nur kurz erwähnt, aber nicht weiterverfolgt. Die bisherigen Ansätze in Wissenschaft und im kommerziellen Bereich sind daher eher revolutionär, d.h. es wird mit speziell entwickelter Software der bestehende Datenbestand analysiert. Danach erfolgt häufig eine einmalige Bereinigung (vor allem bei Einmalprojekten wie z.B. „Data Warehouse"-Projekten) oder es wird ein kontinuierliches Monitoring mit einer ständigen Bereinigung der Daten vorgeschlagen.[30] Entscheidend für eine Vermeidung von Datenqualitätsproblemen sind jedoch die Anwendungssysteme, die diese Daten erzeugen. Ein naheliegender Ansatz könnte darin bestehen, die Anwendungssysteme zu verändern und dadurch die Datenqualität zu verbessern. Dies ist jedoch nicht praktikabel und kann eher als revolutionär und einmalig bezeichnet werden, denn als evolutionär.

Im Rahmen dieser Arbeit wird ein evolutionärer Ansatz entwickelt: Es wird nicht an den Anwendungssystemen angesetzt, sondern am Datenbankmanagementsystem. Dabei liegt der Schwerpunkt auf relationalen Datenbankmanagementsystemen, da diese auf einem wissenschaftlich fundierten Modell basieren und am häufigsten eingesetzt werden. Es wird ein Konzept entwickelt, wie Datenqualitätskomponenten in ein bestehendes RDBMS eingebunden werden können und zwar so, dass Datenqualitätsprobleme bereits bei der Entstehung weitgehend vermieden werden, ohne die auf den Daten basierenden Anwendungssysteme zu verändern. Damit werden Lösungen für Datenqualitätsprobleme an der zentralen Datenhaltungsstelle berücksichtigt, ohne dass jede Anwendung bzw. jede zukünftige Anwendung diese implementieren muss. Mit dieser Lösung sollte es z.B. möglich sein, für eine Datenbanktabelle eine Duplikaterkennung zu aktivieren, um Duplikate inkrementell zu vermeiden, ohne dass an den die Tabelle verwendenden Anwendungen etwas geändert wird. D.h. im ersten Schritt werden Duplikate nur protokolliert, um dann entweder Maßnahmen im Geschäftsprozess vorzunehmen, die diese organisatorisch vermeiden oder die Duplikaterkennung wird aktiviert und während der Erfassung werden Duplikate zurückgewiesen.

[30] Vgl. hierzu: Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006
Helfert, M.: Proaktives Datenqualitätsmanagement in Data-Warehouse-Systemen, Dissertation; logos-Verlag; Berlin; 2002

1.3 Aufbau der Arbeit

Im ersten Kapitel wird zunächst auf die Problematik von Datenqualität und deren Bezug zu Geschäftsprozessen erläutert. Aufgrund bisheriger Ansätze aus der Forschung, die eine Verbesserung bestehender Datenbestände beschreiben, also nachdem die Daten gespeichert sind, soll hier ein Ansatz gefunden werden, der eine Echtzeitvalidierung über ein Datenbankmanagementsystem (vornehmlich relationale DBMS) vorsieht.

Im zweiten Kapitel werden zunächst die Grundlagen von Softwarearchitekturen und darauf aufbauend die allgemein übliche Referenzarchitektur eines Datenbankmanagementsystems beschrieben. Darauf folgen die formalen Grundlagen zur Datenqualität, d.h. es wird geklärt, was Datenqualität eigentlich ist, durch welche Merkmale diese beschreiben wird und welche bekannten Managementansätze zur Verbesserung der Datenqualität existieren.

Kapitel 3 geht dann auf die drei wesentlichen Bausteine einer Datenqualitätskomponente ein: Verstehen, Verbessern und Steuern. Zuvor werden die dafür notwendigen Grundlagen, Methoden und Algorithmen beschrieben, die beim Verstehen, Verbessern und Steuern eingesetzt werden können.

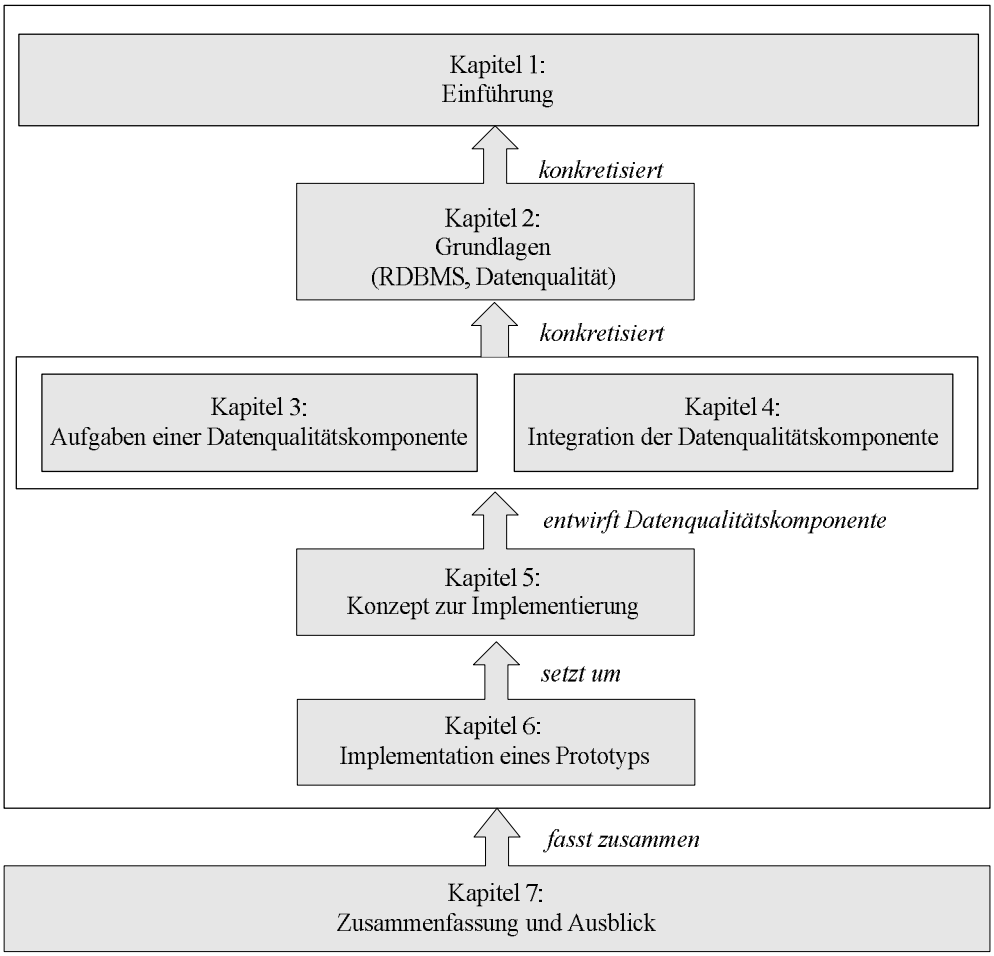
Darauf aufbauend beschreibt und diskutiert Kapitel 4 drei Möglichkeiten zur Einbindung einer Datenqualitätskomponente in ein bestehendes RDBMS. Es wird aufgezeigt, welche dieser Lösungen sich am besten eignet.

Auf Grundlage der Entscheidung aus Kapitel 4 wird in Kapitel 5 ein Konzept für die Architektur der Datenqualitätskomponente im Allgemeinen und die drei Bausteine Verstehen, Verbessern und Steuern im speziellen entworfen.

Kapitel 6 beschreibt schließlich die Umsetzung des Konzeptes in ein bestehendes RDBMS als Prototyp. Dazu wird zunächst die Systemumgebung beschrieben und anschließend die Details des Frameworks in der Implementierung in Form von Klassendiagrammen dargestellt. Einzelne Beispiele zeigen, wie eine evolutionäre Verbesserung von Datenqualität aussieht.

Kapitel 7 schließlich fasst noch einmal die Ergebnisse der Arbeit zusammen, beschreibt kritische Aspekte und gibt einen Ausblick, wie Datenqualität zukünftig in Datenbankmanagementsystemen berücksichtigt werden sollte.

Abb. 4: Gliederung der Arbeit



Quelle: eigene Darstellung

2. Grundlagen

2.1 Softwarearchitektur

Bevor auf relationale Datenbankmanagementsysteme (RDBMS) und die Architektur von Datenbankmanagementsystemen im Allgemeinen eingegangen wird, soll zunächst der Begriff der Softwarearchitektur beschrieben und erklärt werden.

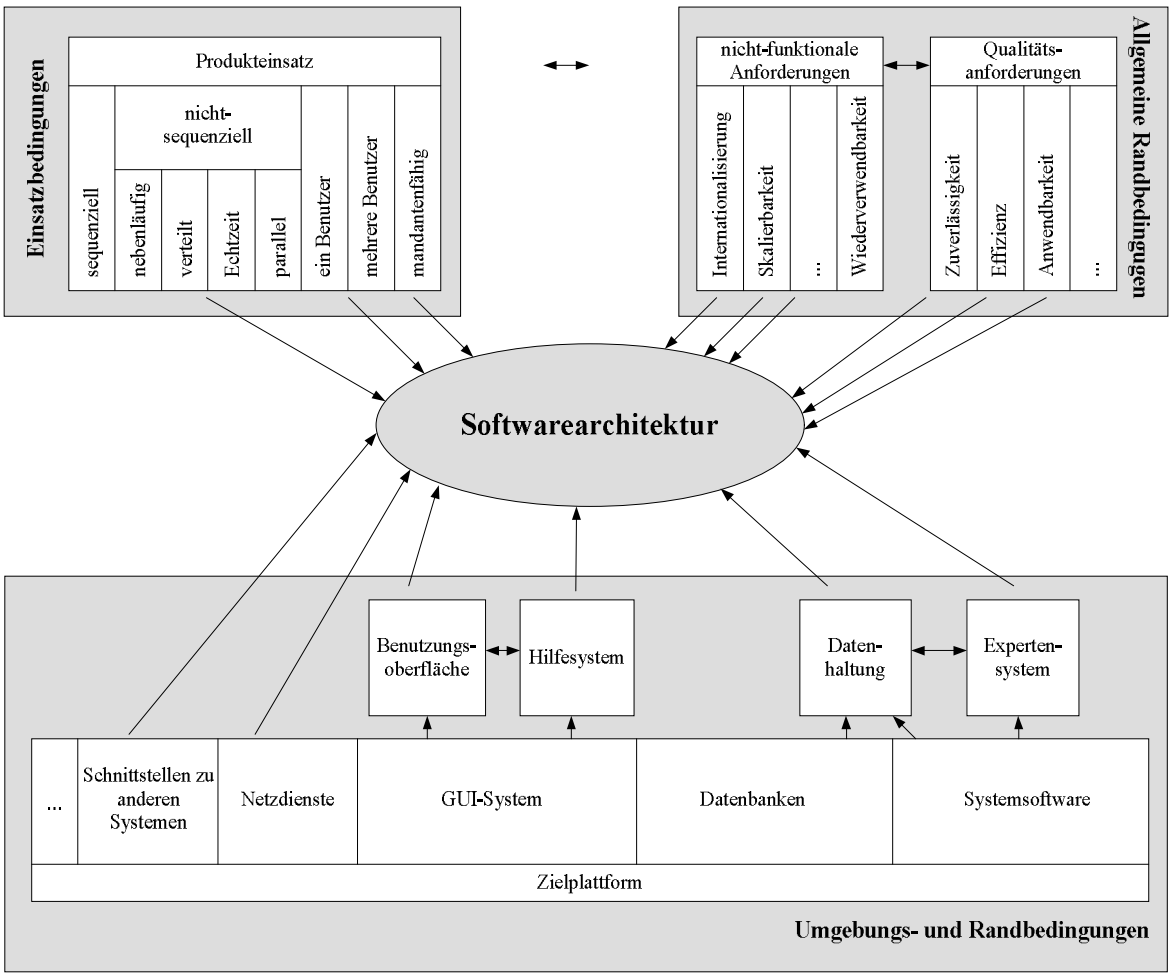
Unter Softwarearchitektur versteht man die Beschreibung der Struktur und Anordnung eines Softwaresystems durch Komponenten und ihren Beziehungen untereinander.[31] Unter den zu beschreibenden Strukturen sind dabei die „wesentlichen und wichtigen Strukturen" des Softwaresystems zu verstehen. Dem Architekten des Softwaresystems obliegt nun die Aufgabe abzugrenzen, welche Strukturen als wesentlich und wichtig eingestuft werden sollten. Dieser Vorgang kann nicht nach formalen Kriterien durchgeführt werden.[32] Da die Softwarearchitektur die Struktur des Softwaresystems beschreibt, handelt es sich nicht um einen detaillierten Entwurf, sondern um eine Beschreibung der Zusammenhänge der Anforderungen an ein zu entwickelndes Softwaresystem.[33] Der Architekt eines Softwaresystems legt zwar die Anordnung und Struktur individuell fest, ist dabei aber von Einflussfaktoren abhängig, die auf die Softwarearchitektur einwirken.[34] Die Priorität, die der Architekt bestimmten Einflussfaktoren zuweist, bestimmt die Struktur und damit letztendlich z.B. wie performant, wartbar oder auch skalierbar das zu entwickelnde Softwaresystem ist (siehe Abb. 5).

Bausteine innerhalb einer Systemarchitektur werden als Komponenten bezeichnet und stellen einen abgegrenzten Teil der Softwarearchitektur dar. Bei sehr vielen Komponenten ist eine stärkere Strukturierung in Form einer Schichten- oder Stufen- bzw. Komponentenarchitektur sinnvoll. Bei der Komponenten- bzw. Stufenarchitektur sind die einzelnen Komponenten lose miteinander gekoppelt, d.h. sie agieren autonom und unabhängig voneinander. Bei der

[31] Balzert, H.: Lehrbuch der Software-Technik - Software-Entwicklung, 2. Auflage; Spektrum Akademischer Verlag; Heidelberg, Berlin; 2000; S. 696
[32] Tabeling, P.: Softwaresysteme und ihre Modellierung; Springer-Verlag; Berlin, Heidelberg; 2006; S. 394f
[33] Reussner, R.; Hasselbring, W.: Handbuch der Software-Architektur; dpunkt Verlag; 2006; Heidelberg; S. 1
[34] Balzert, H.: Lehrbuch der Software-Technik - Software-Entwicklung, 2. Auflage; Spektrum Akademischer Verlag; Heidelberg, Berlin; 2000; S. 687

Schichtenarchitektur sind die Komponenten eng gekoppelt und aufeinander abgestimmt, da höhere Schichten Dienste der darunterliegenden Schichten nutzen.[35]

Abb. 5: Einflussfaktoren einer Softwarearchitektur



Quelle: eigene Darstellung in Anlehnung an: Balzert, H.: Lehrbuch der Software-Technik - Software-Entwicklung, 2. Auflage; Spektrum Akademischer Verlag; Heidelberg, Berlin; 2000; S. 687

Bei der Bildung von Schichten ist die Anzahl der Schichten entscheidend. Werden zu wenig Schichten vorgesehen, so kann das zu Problemen bei der Wiederverwendbarkeit, der Anpassbarkeit und der Portabilität führen. Zu viele Schichten dagegen machen ein Softwaresystem zu komplex und erhöhen meistens den Wartungsaufwand. Wichtig bei der Bildung von Komponenten und Schichten ist zudem die sogenannte Verantwortlichkeit. D.h. Kompo-

[35] Lockemann, P.C.; Dittrich, K.R.: Architektur von Datenbanksystemen; dpunkt Verlag; Heidelberg; 2004; S. 46

nenten sind so zu entwickeln, dass diese die vorher definierten Spezifikationen einhalten[36], die Einhaltung beim Aufruf auch überwachen und dafür verantwortlich sind.

2.2 Relationales Datenbankmanagementsystem (RDBMS)

2.2.1 Einführung

Die Entstehung von Datenbankmanagementsystemen geht auf das Apollo Mondlandeprojekt in den 60er Jahren zurück, da hier viele Daten zu speichern waren, die dateibasiert nicht mehr vernünftig zu verarbeiten waren. In der Folge entwickelte der Hauptauftragnehmer dieses Projektes, die North American Aviation (NAA), die Software GUAM (Generalized Update Access Method), die Daten in einem hierarchischen Strukturbaum speicherte. In einer gemeinsamen Entwicklung zwischen NAA und IBM entstand daraus Mitte der 60er Jahre das noch heute bekannteste hierarchische Datenbankmanagementsystem IMS (Information Management System). Gleichzeitig wurde in dieser Zeit ein anderer Typ eines Datenbanksystems von der Firma General Electric entwickelt, das als Netzwerk Datenbankmanagementsystem bekannt wurde. Dieses DBMS hieß IDS (Integrated Data Store) und konnte im Gegensatz zu hierarchischen Datenbanksystemen komplexere Beziehungen zwischen den Daten abbilden. 1965 wurde auf der CODASYL (Conference on Data Systems Languages) die 1967 umbenannte Data Base Task Group (DBTG) gegründet, der Regierungsvertreter und Vertreter aus der Wirtschaft angehörten. Ziel der DBTG war die Festlegung von Standards für Datenbanken. Wesentliche Nachteile von hierarchischen und Netzwerkdatenbanksystemen waren zum einen der komplexe Programmieraufwand von selbst einfachen Anfragen und das Fehlen einer komfortablen Anfragesprache[37]. Zum anderen fehlten formale theoretischen Grundlagen und vor allem Datenunabhängigkeit[38]. Diese Nachteile führten zur Entwicklung des relationalen Datenmodells und damit zum Erfolg von relationalen gegenüber hierarchischen und Netzwerk-Datenbankmanagementsystemen. So entwickelte E. F. Codd im IBM Forschungslabor das relationale Datenmodell, das 1970 in dem Artikel „A relational model of

[36] Lockemann, P.C.; Dittrich, K.R.: Architektur von Datenbanksystemen; dpunkt Verlag; Heidelberg; 2004; S. 43
[37] Vgl. hierzu: Garcia-Molina, H.; Ullman, J.D.; Widom, J.: Database Systems: The Complete Book; Pearson Education International; Upper Saddle River, New Jersey; 2002; S. 4
[38] Connolly, T.M.; Begg, C.E.: Database Systems, Fourth Edition; Addison-Wesley Pearson Education; Harlow, England; 2005, S. 25

data for large shared data banks“[39] veröffentlicht wurde. Ziel des Modells war ein hoher Grad an Datenunabhängigkeit. Eine Änderung der internen Datenstruktur in der Datenbank sollte nicht automatisch zu einer Modifizierung der auf sie zugreifenden Anwendungen führen.[40]Auf dieser Basis entstand Mitte der 70er Jahre ein erster Prototyp, das System R.[41]

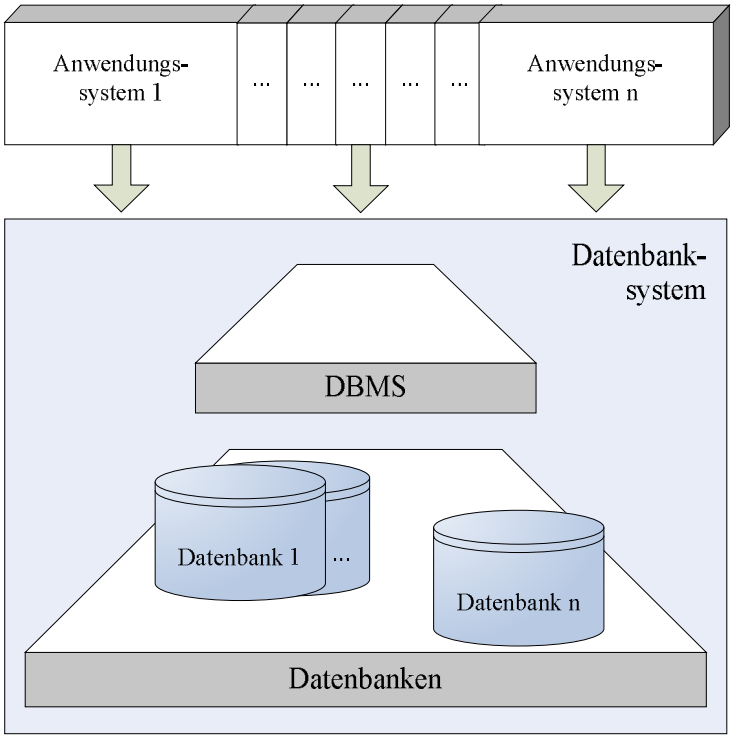
Bevor im weiteren auf die Architektur von relationalen Datenbankmanagementsystemen eingegangen wird, soll zunächst beschrieben werden, was man unter einem Datenbankmanagementsystem versteht und wie die einzelnen Begriffe wie Datenbank, Datenbanksystem und Datenbankmanagementsystem einzuordnen sind.

Unter einer Datenbank versteht man einen integrierten, strukturierten und persistenten Datenbestand. Die Datenbank dient den Benutzern der Daten als Grundlage gemeinsamer Informationen.[42] Sie verwaltet Fakten und Regeln eines Ausschnitts der realen Welt (Miniwelt) in Tabellenform.[43] Unter dem Datenbankmanagementsystem (DBMS) versteht man ein Softwaresystem, über das Datenbanken für unterschiedliche Anwendungssysteme anwendungsneutral und effizient verwaltet werden.[44] Zu den Verwaltungsaufgaben des DBMS gehören neben der Definition und Manipulation von Daten die Sicherstellung von Integritätsbedingungen, Datenunabhängigkeit, Datensicherheit, Datenschutz, Redundanzabbau, sowie die Synchronisation paralleler Zugriffe und die Erstellung von Benutzersichten.[45] Das Datenbanksystem (DBS) schließlich fasst die beiden Komponenten des Datenbankmanagementsystems und der physischen Datenbanken zusammen (siehe Abb. 6).[46]

Anwendungssysteme bilden einen Ausschnitt der realen Welt ab (häufig auch als Miniwelt bezeichnet), indem sie Vorgänge und Sachverhalte dem Anwendungszweck

[39] Codd, E. F.: A Relational Model of Data for Large Shared Data Banks; erschienen in: Communications of the ACM, Vol. 13, Issue 6; New York; 1970; S. 377-387
[40] Connolly, T.M.; Begg, C.E.: Database Systems, Fourth Edition; Addison-Wesley Pearson Education; Harlow, England; 2005, S. 70
[41] Astrahan, M.M. u.a.: System R: Relational Approach to Database Management; erschienen in: ACM Transactions on Database Systems, Vol. 1, No. 2; ACM Press; New York; 1976; S. 97-137
[42] Schneider, M.: Implementierungskonzepte für Datenbanksysteme; Springer Verlag; Berlin, Heidelberg; 2004; S. 3
[43] Türker, C.: SQL:1999 & SQL:2003 - Objektrelationales SQL, SQLJ & SQL/XML; dpunkt Verlag; 2003; S. 2
[44] Schneider, M.: Implementierungskonzepte für Datenbanksysteme; Springer Verlag; Berlin, Heidelberg; 2004; S. 3
[45] Cordts, S.: Datenbankkonzepte in der Praxis; Addison-Wesley; München; 2002; S. 19
[46] Schneider, M.: Implementierungskonzepte für Datenbanksysteme; Springer Verlag; Berlin, Heidelberg; 2004; S. 4

Abb. 6: Datenbanksystem (DBS), Datenbankmanagementsystem (DBMS), Datenbank (DB)



Quelle: eigene Darstellung in Anlehnung an: Cordts, S.: Datenbankkonzepte in der Praxis; Addison-Wesley; München; 2002; S. 20

entsprechend möglichst genau darstellen. Dieser Ausschnitt der realen Welt muss durch ein Modell repräsentiert und abgebildet werden. Die Abbildung der Daten dieser Miniwelt in einem Modell wird als Datenmodell bezeichnet.[47] Das Datenmodell als Grundlage eines DBMS ist ein mathematischer Formalismus mit Notationen zur Beschreibung der Daten und einer Menge von Operationen zur Manipulation dieser Daten.[48] Es dient damit zur Beschreibung eines Ausschnitts der realen Welt in einem „adäquaten Formalismus“[49]. Über diesen Formalismus des Datenmodells werden die Strukturen (Datentypen, Beziehungen, Integritätsbedingungen) der Datenbank beschrieben sowie die Daten abgerufen und manipuliert. Das Datenmodell bestimmt damit letztendlich die „Infrastruktur für die Modellierung der realen Welt“ ähnlich einer Programmiersprache mit Strukturen und Operatoren, die man zur Abbildung der Realwelt benötigt.[50]

[47] Härder, T.: Rahm, E.: Datenbanksysteme - Konzepte und Techniken der Implementierung; Springer Verlag; Berlin, Heidelberg; 1999; S. 3
[48] Schneider, M.: Implementierungskonzepte für Datenbanksysteme; Springer Verlag; Berlin, Heidelberg; 2004; S. 4
[49] Vossen, G.: Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme, 4. Auflage; Oldenbourg Wissenschaftsverlag; München; 2000; S. 17
[50] Kemper, A.; Eickler, A.: Datenbanksysteme, 6. Auflage; Oldenbourg Wissenschaftsverlag; München; 2006; S. 21

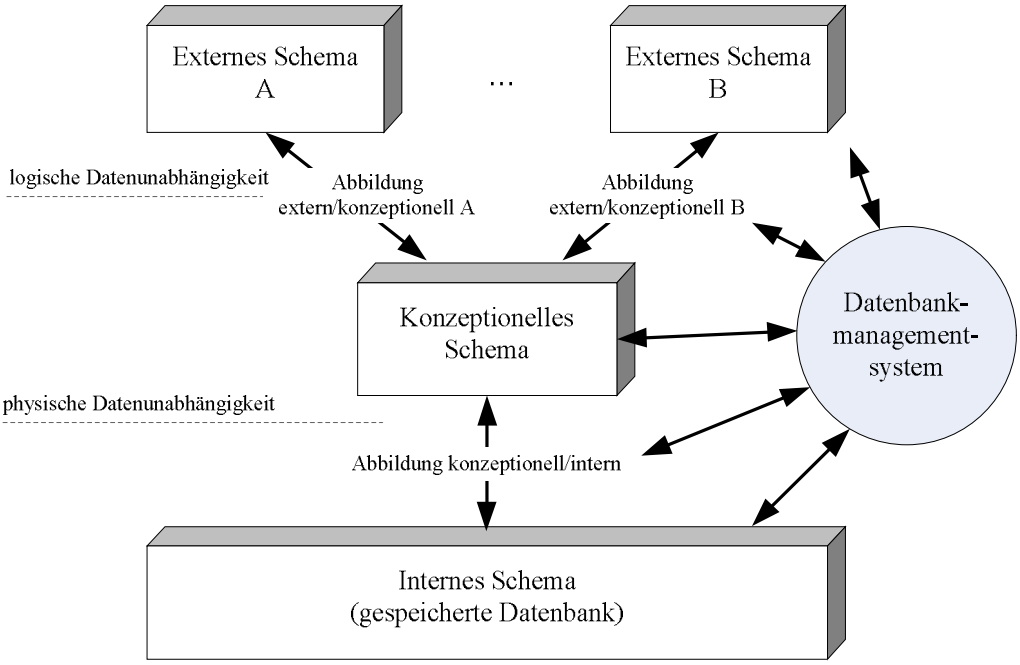
Zu den bekanntesten Datenmodellen gehören das hierarchische, das Netzwerk-, das relationale, und das objektorientierte Datenmodell. Das bekannteste und am weitesten verbreitete Datenmodell, das in heutigen Datenbankmanagementsystemen eingesetzt wird, ist das relationale Datenmodell. Der mathematische Formalismus des Modells basiert auf der Relationenalgebra.

2.2.2 ANSI/SPARC 3-Ebenen Modell

Vor der im nächsten Abschnitt näher zu beschreibenden Referenzarchitektur, die von heutigen DBMS als Grundlage verwendet wird, soll zunächst das stark vereinfachte ANSI/SPARC 3-Ebenen Modell vorgestellt werden, da es als Grundlage für die Referenzarchitektur gilt. Das ANSI/SPARC 3-Ebenen Modell basiert auf einer Architektur von 1971 von der DBTG (Data Base Task Group), die eine 2-Ebenen-Architektur vorschlägt (Systemsicht, als Schema bezeichnet und Benutzersicht, als Subschema bezeichnet).[51] Auf dieser Grundlage und als Folge der Nachteile einer dateibasierten Speicherung[52] entstand 1975 vom „American National Standards Institute“ (ANSI) „Standards Planning amd Requirements Committee“ (SPARC) der Vorschlag für die abstrakte Architektur eines DBMS. Ziel war es, die Anwendersicht auf die Daten und die physische Sicht zu trennen. Erreicht wird dieses Ziel durch eine 3-Teilung in ein externes Schema (Sichten), ein konzeptionelles und ein internes (physisches) Schema. Hierdurch soll das Konzept der Datenunabhängigkeit erreicht werden. Die Änderung des Schemas einer Ebene führt also nicht zwangsläufig dazu, dass das Schema einer anderen Ebene geändert werden muss.[53]

[51] Connolly, T.M.; Begg, C.E.: Database Systems, Fourth Edition; Addison-Wesley Pearson Education; Harlow, England; 2005; S. 35
[52] Vgl. hierzu: Cordts, S.: Datenbankkonzepte in der Praxis; Addison-Wesley; München; 2002; S. 18ff
Connolly, T.M.; Begg, C.E.: Database Systems, Fourth Edition; Addison-Wesley Pearson Education; Harlow, Enland; 2005; S. 7ff
[53] Elmasri, R.; Navathe, S. B.: Grundlagen von Datenbanksystemen; 3. Auflage, Pearson Education; München; 2005; S. 45

Abb. 7: Schnittstellen und ihre Abbildung im ANSI/SPARC-Vorschlag



Quelle: eigene Darstellung in Anlehnung an: Härder, T.: Rahm, E.: Datenbanksysteme - Konzepte und Techniken der Implementierung; Springer Verlag; Berlin, Heidelberg; 1999; S. 10 und Connolly, T.M.; Begg, C.E.: Database Systems, Fourth Edition; Addison-Wesley Pearson Education; Harlow, Enland; 2005; S. 39

Soll ein Anwendungssystem entwickelt werden, so wird ein Ausschnitt der realen Welt in Form eines Datenmodells modelliert. Der Entwurf der Datenbank orientiert sich dabei zunächst an einer gemeinschaftlichen Sicht der Daten, dem konzeptionellen Schema.[54] Das konzeptionelle Schema legt also fest, welche Daten abgespeichert werden und ist ein Abbild der gesamten Informationsmenge. Das externe Schema unterteilt das konzeptionelle wiederum in einzelne Schemata, die den Bedürfnissen einer Anwendung bzw. einer Benutzergruppe entsprechen.[55] Das physische bzw. interne Schema bestimmt schließlich, wie die Daten persistent im Hintergrundspeicher gespeichert werden (siehe Abb. 7).

Da jeder Anwender im 3-Ebenen-Modell auf sein eigenes externes Schema zugreift, müssen zwischen den Ebenen Transformationen durchgeführt werden. Bei einer Abfrage muss zunächst das externe Schema auf das konzeptionelle und dieses dann auf das physische

[54] Härder, T.: Rahm, E.: Datenbanksysteme - Konzepte und Techniken der Implementierung; Springer Verlag; Berlin, Heidelberg; 1999; S. 10
[55] Kemper, A.; Eickler, A.: Datenbanksysteme, 6. Auflage; Oldenbourg Wissenschaftsverlag; München; 2006; S. 20

abgebildet werden. Diese Transformationsvorgänge werden als Abbildungen bzw. Mappings bezeichnet.[56]

Von den meisten Datenbankmanagementsystemen wird die Trennung in drei Ebenen in der Architektur nicht vollständig umgesetzt, so sind z.B. Eigenschaften der physischen Ebene in der konzeptionellen Ebene enthalten.

2.2.3 Referenzarchitektur eines DBMS

Das im Folgenden beschriebene 5-Schichten-Modell ist angelehnt an die von T. Härder und A. Reuter 1985 beschriebene Referenzarchitektur, die die Grundlage heutiger Datenbankmanagementsysteme bildet.[57] Wie im Abschnitt über Softwarearchitekturen erläutert, wird die Architektur eines Softwaresystems von unterschiedlichen Einflussfaktoren geprägt. Die Architektur eines DBMS ist stark geprägt von zwei Kern-Anforderungen: Der Datenunabhängigkeit und der Performanz.

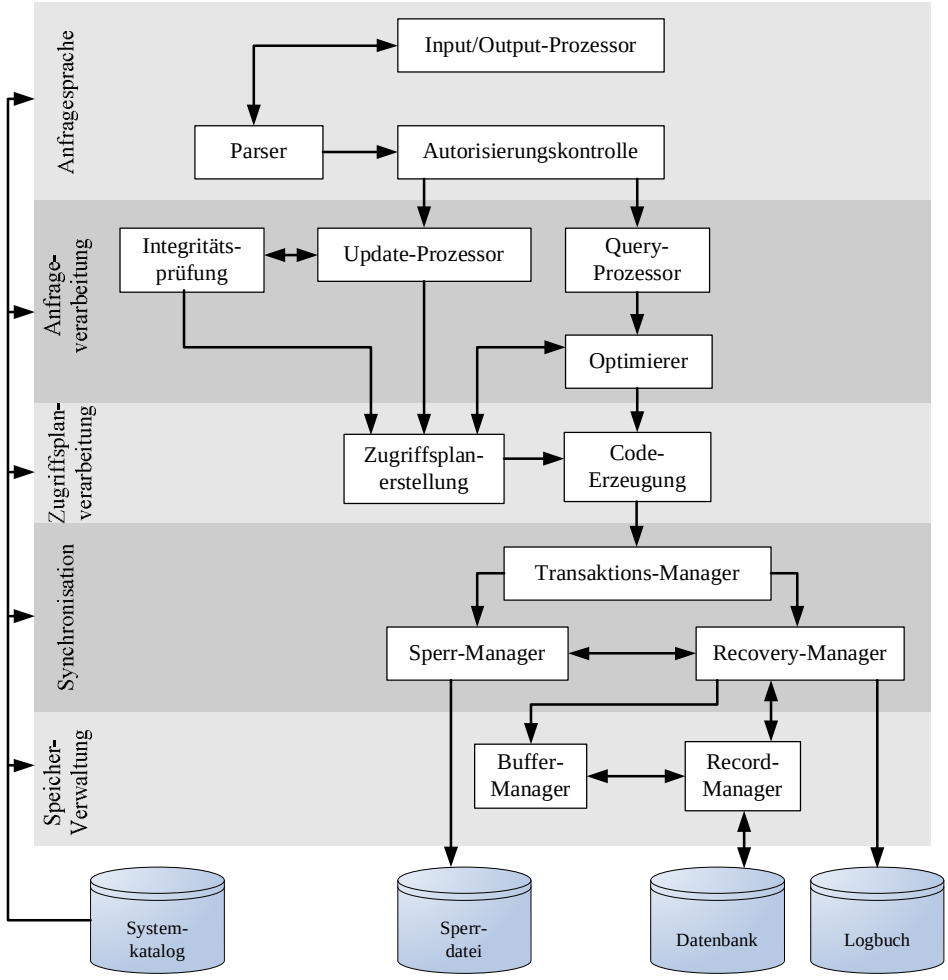
Eine Lösung für die Anforderung der Datenunabhängigkeit war das Hauptziel des in Abschnitt 2.2.2 beschriebenen ANSI/SPARC-Modells. Zu diesem Zweck wird das DBMS in drei Schichten unterteilt, wodurch Änderungen am konzeptionellen und vor allem am physischen Schema keinen unmittelbaren Einfluss auf die darüberliegenden Anwendungssysteme haben. Performanz wiederum ist stark abhängig von dem darunterliegenden Computersystem, auf dem das DBMS läuft.[58] Eine DBMS-Architektur, die Performanz als vorrangiges Ziel erreichen will, muss zwangsläufig vor allem Hardware- und Betriebssystemeigenschaften berücksichtigen, d.h. die strukturelle Speicherung von Daten in Sekundärspeichern muss in der Architektur reflektiert werden (Anzahl und Größe von Spuren, Sektoren und Blöcken).[59] Weitere wichtige Architekturanforderungen an ein DBMS sind außerdem: Effizienter Datenzugriff, gemeinsame Datenbasis, gleichzeitiger bzw. nebenläufiger Daten-

[56] Elmasri, R.; Navathe, S. B.: Grundlagen von Datenbanksystemen, 3. Auflage; Pearson Education; München; 2005; S. 44f
[57] Härder, T.: DBMS Architecture - the Layer Model and its Evolution; erschienen in: Datenbank-Spektrum, 5. Jahrgang, Heft 13; dpunkt Verlag; Heidelberg; 2005; S. 46
[58] Silberschatz, A.; Korth, H.F.; Sudarshan, S.: Database System Concepts, Fifth Edition; McGraw-Hill; New York; 2006; S. 783
[59] Schneider, M.: Implementierungskonzepte für Datenbanksysteme; Springer Verlag; Berlin, Heidelberg; 2004; S. 20

zugriff, Redundanzvermeidung, Datenkonsistenz, Datenintegrität, Datensicherheit, Backup/ Recovery, Abfragen und Abfragesprachen, Flexibilität.[60]

Um Performanz zu erreichen, ist die Referenzarchitektur als eng gekoppelte Schichtenarchitektur (siehe Abb. 8) entworfen und in den heutigen DBMS auch grundsätzlich so implementiert.

Abb. 8: Schichtenarchitektur für Datenbankmanagementsysteme



Quelle: eigene Darstellung in Anlehnung an: Vossen, G.: Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme, 4. Auflage; Oldenbourg Wissenschaftsverlag; München; 2000; S. 33

Die Referenzarchitektur eines Datenbankmanagementsystems besteht aus fünf Schichten. Dabei unterscheiden sich die einzelnen Schichten in der Granularität der Daten und im

[60] Schneider, M.: Implementierungskonzepte für Datenbanksysteme; Springer Verlag; Berlin, Heidelberg; 2004; S. 8

Abstraktionsgrad des Zugriffs von der untersten Schicht, der Speicherverwaltung, bis zur obersten Schicht, der Anfragesprachenverarbeitung.[61]

Die unterste Schicht der Speicher- oder Geräteverwaltung (Disk Space Manager) verwaltet die Hardwareressourcen und damit alle physischen Zugriffe zwischen dem Sekundärspeicher, also den externen Speichermedien, und dem Primärspeicher, dem Hauptspeicher. Die Schicht dient der Bereitstellung und Freigabe von Speicher und dem Auffinden eines Datenobjektes auf Basis von Dateien und Blöcken. Über die Systempufferverwaltung (Buffer Manager) wird der zur Verfügung stehende Puffer (Hauptspeicher) in sogenannte Seiten (Pages), die normalerweise der Größe eines Blockes entsprechen, unterteilt. Dadurch werden die Zugriffe auf die externen Speichermedien minimiert. Wird ein Datenbankobjekt über die darüberliegenden Schichten angefordert, das noch nicht im Systempuffer gespeichert ist, so stellt die Systempufferverwaltung freien Platz hierfür zur Verfügung.[62] Nicht mehr benötigte Seiten werden für neu angeforderte Seiten aus dem Hauptspeicher durch sogenannte Seitenwechselstrategien entfernt[63] und bei einer Veränderung in den externen Speicher zurückgeschrieben.[64]

Über die Synchronisationsverwaltung werden parallele Zugriffe durch mehrere Benutzer auf die Datenbank synchronisiert, da eine Datenbank normalerweise nicht nur einem einzigen Benutzer exklusiv zur Verfügung steht. Über den Transaktions-Manager werden parallel ablaufende Transaktionen synchronisiert. Bei einer Transaktion handelt es sich um eine Folge von Anweisungen, die nur gesamt oder überhaupt nicht, d.h. atomar, ausgeführt werden.[65] Ein sequentieller Ablauf von Transaktionen ist aus naheliegenden Gründen nicht sinnvoll, daher existieren Strategien zur parallelen Abarbeitung (Concurrency Control). Dazu werden vom Sperr-Manager Sperren (Locks) auf Datenobjekte gesetzt, die bei einem parallelem Zugriff berücksichtigt werden müssen. Diese Sperren werden vom Sperr-Manager verwaltet und in einer Sperr-Datei (Lock File) gespeichert. Man spricht in diesem Zusammenhang von der Isolation von Transaktionen im Mehrbenutzerbetrieb. Wurden Transaktionen nicht erfolgreich

[61] In der Literatur findet man Abwandlungen dieser Architektur, z.B. nur 3 Schichten, die grundlegende Struktur und die wesentlichen Komponenten sind aber i.d.R. identisch
[62] Härder, T.: Rahm, E.: Datenbanksysteme - Konzepte und Techniken der Implementierung; Springer Verlag; Berlin, Heidelberg; 1999; S. 107
[63] Saake, G.; Heuer, A.; Sattler, K.-U.: Datenbanken: Implementierungstechniken, 2. Auflage; mitp-Verlag; Bonn; 2005; 46
[64] Härder, T.: Rahm, E.: Datenbanksysteme - Konzepte und Techniken der Implementierung; Springer Verlag; Berlin, Heidelberg; 1999; S. 107
[65] Kemper, A.; Eickler, A.: Datenbanksysteme, 6. Auflage; Oldenbourg Wissenschaftsverlag; München; 2006; S. 269

durchgeführt, so ist es die Aufgabe des Recovery-Managers die Datenbank wieder in den Zustand vor der Transaktion zu setzen. Dazu protokolliert er alle Änderungen an der Datenbank in einem Logbuch. Auch bei unerwarteter Beendigung des DBMS durch Hard- oder Softwarefehler ist der Recovery-Manager für die konsistente widerspruchsfreie Herstellung der Datenbank zuständig.

Die Schicht der Zugriffsplanverarbeitung (Access Path Manager) verwaltet externe Datenstrukturen zum Speichern und zum Zugriff auf Datensätze (Indexstrukturen). Werden über die Anfragesprache Daten abgefragt, so ist es die Aufgabe der Zugriffsplanverarbeitung über die zur Verfügung stehenden Zugriffstrukturen einen möglichst effizienten Zugriffsplan zu erstellen und hierfür Code zu generieren.[66]

Über die Schicht der Anfrageverarbeitung werden vom Anwender gestellte Anfragen, die in der obersten Schicht vorbereitet wurden, vom Query-Prozessor abgearbeitet. Dieser ermittelt mit dem Anfrage-Optimierer (Query Optimizer) einen effizienten Ausführungsplan mit Hilfe eines Kostenmodells, das auf der Basis statistischer Werte, Schema-, Indexinformationen und Attributverteilungen erstellt wird.[67] Änderungsanfragen werden vom Update-Prozessor bearbeitet, um mit dem Integritäts-Manager (Integrity Manager) vorhandene Integritätsbedingungen (Integrity Constraints) zu überprüfen. Integritätsbedingungen dienen der semantischen Korrektheit der Datenbank[68] und damit der Sicherstellung von Datenqualitätsaspekten der in ihr gespeicherten Daten.

Die Anfrageverarbeitung erhält die vom Anwender gestellten Anfragen und analysiert diese syntaktisch über den Parser,[69] der die Anfrage intern in einen Syntaxbaum (Ausdruck in relationaler Algebra)[70] umwandelt. In relationalen Datenbankmanagementsystemen wird als Anfragesprache das deskriptive SQL (Structured Query Language) verwendet. Über die Autorisierungskontrolle erfolgt eine Zugriffskontrolle, ob der Anwender auf die Daten zugreifen darf und hierfür die nötigen Berechtigungen besitzt. Informationen hierüber werden

[66] Vossen, G.: Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme, 4. Auflage; Oldenbourg Wissenschaftsverlag; München; 2000; S. 34
[67] Kemper, A.; Eickler, A.: Datenbanksysteme, 6. Auflage; Oldenbourg Wissenschaftsverlag; München; 2006; S. 233
[68] Vossen, G.: Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme, 4. Auflage; Oldenbourg Wissenschaftsverlag; München; 2000; S. 32
[69] Kemper, A.; Eickler, A.: Datenbanksysteme, 6. Auflage; Oldenbourg Wissenschaftsverlag; München; 2006; S. 233
[70] Silberschatz, A.; Korth, H.F.; Sudarshan, S.: Database System Concepts, Fifth Edition; McGraw-Hill; New York; 2006; S. 532

in der Systemkatalog-Datenbank gespeichert. Der Input/Outputprozessor nimmt die Anfragen des Anwenders entgegen und liefert ein Ergebnis bzw. eine Fehlermeldung zurück.

Verwaltungsdaten, die von den einzelnen Schichten geschrieben und gelesen werden, sind in der Systemkatalog-Datenbank gespeichert (System Catalog, Data Dictionary). Diese Datenbank speichert u.a. die Definition der Struktur der Datenbanken (Metadaten), dazugehörige Verwaltungsinformationen (z.B. Zugriffsrechte von Benutzern), Integritätsbedingungen, Informationen über die Indexstrukturen (z.B. Dateiname eines B+-Baumes usw.) und statistische Informationen über die Daten für den Query Optimizer.

Die Referenzarchitektur heutiger Datenbankmanagementsysteme hat sich in den letzten 30 Jahren wenig verändert,[71] obwohl es viele Veränderungen bzgl. Skalierbarkeit, Performanz und Funktionalität gegeben hat. Dass diese Änderungen durch die Referenzarchitektur abgebildet werden konnten, ist ein Indiz für ein gutes Datenmodell[72] und eine für die neu entstandenen Anforderungen geeignete Referenzarchitektur.

2.2.4 Relationale Datenbanken

Nachdem in letzten beiden Abschnitten zunächst erläutert wurde, was unter Datenbankmanagementsystemen zu verstehen ist und wie die Referenzarchitektur einer Implementierung eines DBMS aussieht, wird nun auf das relationale Datenmodell respektive auf relationale Datenbanken im Speziellen eingegangen.

Im relationalen Datenmodell existiert eine stringente Festlegung der Struktur der Daten, weshalb das Datenmodell auch nur die Speicherung strukturierter Daten vorsieht. Semistrukturierte Daten sind nur ansatzweise durch XML seit der Standardisierung der Anfragesprache SQL:2003 und durch objektrelationale Eigenschaften verwaltbar, werden aber nur bedingt unterstützt und hier auch nicht weiter betrachtet.

Das relationale Datenmodell basiert formal auf der Relationenalgebra. Objekttypen, also Abbildungen von Objekten der realen Welt (z.B. Kunde), werden deshalb im relationalen

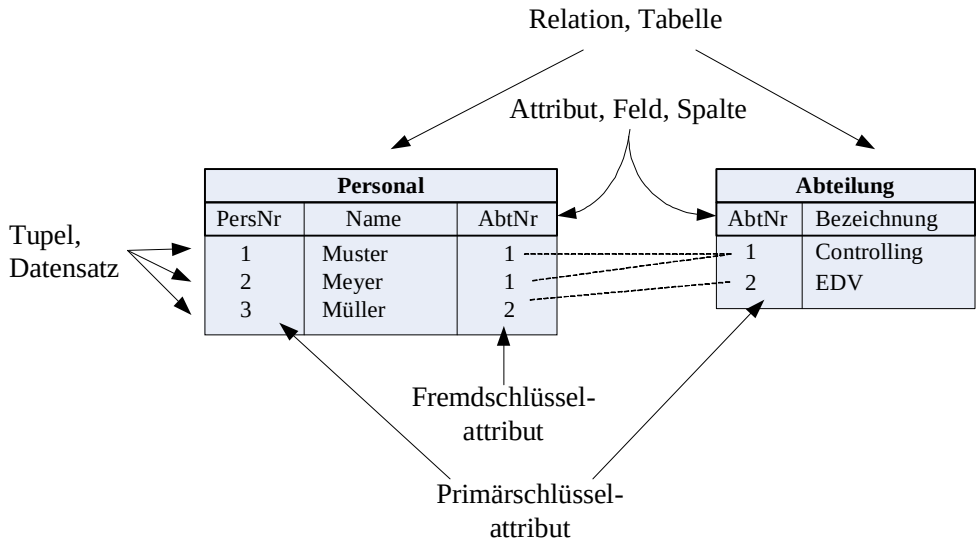
[71] Lockemann, P.C.; Dittrich, K.R.: Architektur von Datenbanksystemen; dpunkt Verlag; Heidelberg; 2004; S. 369
[72] Härder, T.: DBMS Architecture - the Layer Model and its Evolution; erschienen in: Datenbank-Spektrum, 5. Jahrgang, Heft 13; dpunkt Verlag; Heidelberg; 2005; S. 49

Datenmodell in Form von Mengen bzw. Relationen dargestellt. Objekte eines Objekttyps werden durch Attribute beschrieben und als Tupel gespeichert. Da es sich um Mengen handelt, besitzen weder die Objekte in den Relationen noch die Attribute selbst eine Sortierung oder eine bestimmte Reihenfolge. Zur Vereinfachung werden Relationen normalerweise dennoch als 2-dimensionale Tabellen abgebildet. Allgemein werden deshalb die Begriffe Relation, Tupel und Attribut eher selten verwendet. Vielmehr spricht man von Tabelle (Table), Datensatz (Record, Row) und Spalte (Datenfeld, Field, Column). Im Folgenden werden diese Begriffe daher synonym verwendet.

Innerhalb einer Tabelle werden Datensätze eindeutig durch ihren Primärschlüssel gekennzeichnet, der aus einer oder mehreren Spalten bestehen kann, aber immer minimal in der Anzahl der Spalten sein muss. Der Wert des Primärschlüssels muss eindeutig und darf nicht NULL sein, d.h. er muss bekannt sein. Um nun Beziehungen zwischen Objekttypen darzustellen, wird im Relationenmodell der Begriff des Fremdschlüssels eingeführt. Eine Fremdschlüsselspalte ist eine Spalte, deren Wert einen Bezug zu einem Primärschlüssel einer anderen Tabelle herstellen soll. Angenommen, man hat eine Tabelle Mitarbeiter und eine Tabelle Abteilung. Die Beziehung, dass jeder Mitarbeiter zu einer Abteilung gehört, kann abgebildet werden, indem die Tabelle Mitarbeiter den Primärschlüssel der Tabelle Abteilung als sogenannten Fremdschlüssel übernimmt. Diese Fremdschlüsselspalte darf dann nur solche Werte enthalten, die auch in der korrespondierenden Primärschlüsselspalte vorkommen. Diese Einschränkung wird als referentielle Integrität oder Fremdschlüsselintegrität bezeichnet und legt fest, dass der Wert eines Fremdschlüssels immer einem Wert der dazugehörigen Primärschlüsselspalte entsprechen muss.[73]

[73] Vgl. hierzu: Cordts, S.: Datenbankkonzepte in der Praxis; Addison-Wesley; München; 2002; S. 72f

Abb. 9: Elemente des relationalen Datenmodells



Quelle: eigene Darstellung

Seit den Anfängen relationaler Datenbankmanagementsysteme haben sich diese bis heute immer mehr von einer reinen Datenhaltung im Sinne des relationalen Datenmodells zu hochkomplexen Applikationsservern entwickelt. Mit der Einsicht, dass Daten nicht von Algorithmen getrennt werden können, entstanden die objektorientierten Konzepte, die sich in objektrelationalen Konstrukten in relationalen Datenbankmanagementsystemen wiederfinden. Hinzu kommen weitere Sprachkonstrukte wie Trigger, Stored Procedures, Funktionen, User Defined Types, neue Datentypen mit deren Hilfe die Geschäftslogik in ein RDBMS abgebildet werden kann. Neben eigentlichen Sprachkonstrukten zur programmtechnischen Abbildung von Geschäftslogik enthalten moderne RDBMS auch immer mehr lose gekoppelte Komponenten, die andere Aufgaben der Datenverwertung und -bereitstellung lösen. Hierzu gehören z.B. ETL-, OLAP-, Data Mining-, Webservicebereitstellungs-, Workflow-, asynchrone Queuekomponenten. Mehr und mehr lösen also heute RDBMS Aufgaben, die vorher noch von getrennten externen Anwendungssystemen übernommen wurden, da sich eine Integration solcher Anwendungsaufgaben als sinnvoll erweist.[74]

Das relationale Datenmodell als solches berücksichtigt als Datenqualitätsaspekt indirekt nur die Einschränkung der referentiellen Integrität und das auch nur formal. Viele der neuen

[74] Vgl. hierzu: Gray, J.: The Revolution in Database Architecture, Keynote talk; erschienen in: ACM SIGMOD 2004; Paris; 2004 und
Dittrich, K.R.; Geppert, A.: Component Database Systems: Introduction, Foundations, and Overview; erschienen in: Dittrich, K.R.; Geppert, A. (Eds.): Component Database Systems; Morgan Kaufmann; San Francisco; 2001; S. 2

Anforderungen an ein RDBMS spiegeln sich vielmehr in der Standardisierung der relationalen Anfragesprache, der Structured Query Language (SQL), wider, die im Folgenden unter Berücksichtigung von Integritätsbedingungen näher betrachtet werden soll.

2.2.5 Relationale Anfragesprache

2.2.5.1 Einführung

Mit der Entwicklung des ersten RDBMS-Prototypen von IBM zu Beginn der 70er Jahre, dem „System R“, wurde auch eine Anfragesprache mit dem Namen SEQUEL (Structured English Query Language) entwickelt.[75] Aus einer Untermenge des Nachfolgers SEQUEL2 entstand dann SQL (Structured Query Language).[76] Der Prototyp „System R“ bildete schließlich die Grundlage des kommerziellen RDBMS SQL/DS von IBM. Weitere bekannte relationale Datenbankmanagementsysteme folgten mit dem Oracle Enterprise Server von Oracle, DB2 von IBM und Microsoft SQL Server. Daneben existieren bekannte Open-Source-Produkte wie MySQL oder PostGreSQL.[77]

Kernziel bei der Architektur eines DBMS ist vor allem die Anforderung der Datenunabhängigkeit. Anfragen werden deshalb auf dem logischen Schema formuliert, um nicht von der physischen Speicherstruktur abhängig zu sein.[78] Dieses spiegelt sich auch im standardisierten SQL wider. So gibt es z.B. bis heute keine Standardisierung zum Anlegen von Indizes auf Tabellen.[79]

SQL wurde 1986 das erste Mal durch das ANSI (American National Standards Institute) standardisiert. 1989 wird eine revidierte Fassung von SQL-86 von der ISO (International Standardization Organization) als SQL-89 veröffentlicht.[80] Es folgte 1992 die erste gemeinsame Standardisierung von ANSI und ISO, der SQL-92 oder SQL-2 Standard. 1999

[75] Vgl. hierzu: Kemper, A.; Eickler, A.: Datenbanksysteme, 6. Auflage; Oldenbourg Wissenschaftsverlag; München; 2006; S. 107
Chamberlin, D.D.; Boyce, R.F.: SEQUEL: Structured English Query Language; erschienen in: Proceedings of the 1974 ACM SIGFIDET workshop on Data description, access and control: Data models: Data-structure-set versus relational; Ann Arbor, Michigan; 1974; S. 249-263
[76] Türker, C.: SQL:1999 & SQL:2003 - Objektrelationales SQL, SQLJ & SQL/XML; dpunkt Verlag; 2003; S. 1
[77] Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 19
[78] Kemper, A.; Eickler, A.: Datenbanksysteme, 6. Auflage; Oldenbourg Wissenschaftsverlag; München; 2006; S. 233
[79] die bekannten RDBMS verwenden hier allerdings eine ähnliche Syntax
[80] Türker, C.: SQL:1999 & SQL:2003 - Objektrelationales SQL, SQLJ & SQL/XML; dpunkt Verlag; 2003; S. 2

wird SQL 3 mit der offiziellen Bezeichnung SQL:1999 veröffentlicht, das als wesentliche Erweiterungen objektrelationale Elemente und Trigger enthält. SQL:2003 schließlich besitzt geringfügige Erweiterungen gegenüber SQL:1999.[81] Hierzu gehört als wesentliches Merkmal die XML-Integration, daneben werden u.a. Mehrfachmengen (Multiset), Identitätsspalten und tabellenwertige Funktionen unterstützt[82].

SQL:2003 besteht aus folgenden Teilen:[83]

Part 1: SQL/Framework gibt einen Überblick über den Standard und dessen Sprachebenen.

Part 2: SQL/Foundation beschreibt die Grundlagen, also das Datenmodell von SQL, Datendefinition, Datenabfrage und Datenmanipulation. Hierzu gehören Tabellen, Sichten, Typen, Schemata, Abfragen und Änderungen, Ausdrücke, Prädikate, Sicherheitsaspekte, Transaktionen usw.

Part 3: SQL/CLI (Call Level Interface) legt eine API (Application Programming Interface) von niedrigen Schnittstellen für den Zugriff von Anwendungen auf Datenbanken fest.

Part 4: SQL/PSM (Persistent Stored Modules) legt prozedurale Sprachelemente und die Einbindung anderer Programmiersprachen fest.

Part 9: SQL/MED (Management of External Data) legt fest, wie auf externe Daten zugegriffen werden kann.

Part 10: SQL/OLB (Object Language Binding) beschreibt das Einbetten von SQL in Java.

Part 11: SQL/Schemata (Information and Definition Schema) legt den Standard für Metatabellen fest.

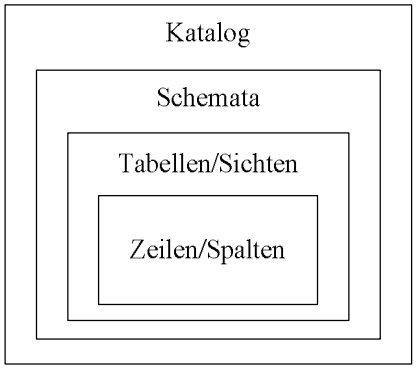
Part 13: SQL/JRT (Java Routines and Types) legt fest wie auf Routinen und Typen, die in Java definiert wurden, zugegriffen wird.

Part 14: SQL/XML beschreibt Spezifikationen zur Einbindung von XML in SQL.

Für fehlende Teilnummern existiert zurzeit noch kein verabschiedeter Standard.[84] Nach dem SQL-Standard wird die DBMS Umgebung in Kataloge (die jeweils den einzelnen Datenbanken entsprechen) sowie in Schemata, Tabellen/Sichten und Zeilen/Spalten unterteilt.

[81] Silberschatz, A.; Korth, H.F.; Sudarshan, S.: Database System Concepts, Fifth Edition; McGraw-Hill; New York; 2006; S. 896
[82] Türker, C.: SQL:1999 & SQL:2003 - Objektrelationales SQL, SQLJ & SQL/XML; dpunkt Verlag; 2003; S. 12
[83] Silberschatz, A.; Korth, H.F.; Sudarshan, S.: Database System Concepts, Fifth Edition; McGraw-Hill; New York; 2006; S. 896
[84] Silberschatz, A.; Korth, H.F.; Sudarshan, S.: Database System Concepts, Fifth Edition; McGraw-Hill; New York; 2006; S. 896f

Abb. 10: Umgebung nach SQL-Standard



Quelle: eigene Darstellung in Anlehnung an: Melton, J.; Simon, A.R.: SQL:1999 - Understanding Relational Language Components; Morgan Kaufmann; San Francisco; 2002; S. 84

Datenbankobjekte wie Tabellen usw. werden in einem Schema gespeichert. Mehrere Schemata wiederum werden in einem Katalog zusammengefasst. Schemaobjekte werden wie folgt referenziert:[85]

<catalog name>.<schema name>.<object name>

Im Folgenden wird zunächst auf Sprachelemente, die zur Sicherstellung semantischer Integritätsbedingungen im SQL-Standard vorhanden sind, eingegangen, da diese originär zur Sicherstellung der Datenqualität eingesetzt werden können. Danach werden prozedurale Sprachelemente beschrieben, da viele der Algorithmen zur Datenqualität nur über eine mehr oder weniger komplexe Programmierung zu realisieren sind. Es ist es deshalb notwendig zu entscheiden, inwieweit diese Sprachelemente zur Implementierung einer Datenqualitäts-komponente geeignet sind oder ob es vorteilhafter ist, auf externe, in einer anderen Programmiersprache geschriebene Komponenten zuzugreifen.

[85] Melton, J.; Simon, A.R.: SQL:1999 - Understanding Relational Language Components; Morgan Kaufmann; San Francisco; 2002; S. 86

2.2.5.2 Semantische Integritätsbedingungen

Datenbankmanagementsysteme sollen nicht nur der einfachen Verarbeitung von Daten dienen, sondern auch die Konsistenz, also die Widerspruchsfreiheit der Daten sicherstellen. Hierzu dienen semantische Integritätsbedingungen. Sie sollen nur semantisch korrekte Datenbankzustände zulassen.[86] Unter semantischen Integritätsbedingungen sind Einschränkungen zu verstehen, die die Plausibilität von Daten überprüfen und davon abhängig festlegen, ob die Daten geändert, gelöscht oder eingefügt werden sollen.[87] Die Bedeutung von Integritätsbedingungen in Datenbanken wurde bereits in den 70er Jahren von E. F. Codd, M. Stonebraker u.a. in verschiedenen Artikeln erkannt.[88]

Integritätsbedingungen werden der abzubildenden Realwelt entnommen und können durch Regeln abgebildet werden. Sind Integritätsbedingungen im DBMS integriert, muss nicht jede Anwendung, auf dessen Einhaltung achten. Das führt zu einer wesentlichen Vereinfachung der Wartung und einer geringeren Fehleranfälligkeit.[89]

E. F. Codd unterscheidet bei Integritätsbedingungen zwischen Entitätintegrität, referentieller Integrität und Spaltenintegrität. Daneben führt er eine allgemein verwendbare Kategorie ein: Geschäftsregeln. Gegenüber den Geschäftsregeln können die anderen Integritätsbedingungen als kontextunabhängig betrachtet werden. Entitätenintegrität bezieht sich auf die Einschränkung des Primärschlüssels, der nicht NULL sein darf und eindeutig sein muss. Referentielle Integrität legt fest, dass der Wert eines Fremdschlüssels mit dem eines Primärschlüssels einer Tabelle übereinstimmen muss oder NULL sein darf. Mit Spaltenintegrität ist die Einschränkung einer Spalte auf bestimmte Werte gemeint.[90]

A. Kemper und A. Eickler unterscheiden zwischen statischen und dynamischen Integritätsbedingungen.[91] Statische Integrität betrifft die Wertebeschränkung bezogen auf eine oder

[86] Türker, C.: SQL:1999 & SQL:2003 - Objektrelationales SQL, SQLJ & SQL/XML; dpunkt Verlag; 2003; S. 19
[87] Vgl. hierzu: Hausotter, A.: Datenorganisation; erschienen in: Disterer, G.; Fels, F.; Hausotter, A.: Taschenbuch der Wirtschaftsinformatik, 2. Auflage; Carl Hanser Verlag; München, Wien; 2003; S. 212
[88] Türker, C.; Gertz, M.: Semantic integrity support in SQL:1999 and commercial (object-)relational database management systems; erschienen in: The VLDB Journal - The international Journal on Very Large Data Bases, Vol. 10, Issue 4; Springer Verlag; New York; 2001; S. 241
[89] Kemper, A.; Eickler, A.: Datenbanksysteme, 6. Auflage; Oldenbourg Wissenschaftsverlag; München; 2006; S. 155
[90] Vgl. hierzu: Lee, Y.W.; Pipino, L.L.; Funk, J.D.; Wang, R.Y.: Journey to Data Quality; MIT Press; Cambridge, Massachusetts; 2006; S. 53
[91] Kemper, A.; Eickler, A.: Datenbanksysteme, 6. Auflage; Oldenbourg Wissenschaftsverlag; München; 2006; S. 155

mehrere Tabellen und dynamische Integrität den Werteübergang (z.B. vom Wert „ledig“ nur Übergang auf „verheiratet“).[92]

Werden Daten geändert, eingefügt oder gelöscht und dabei eine oder mehrere Integritätsbedingungen verletzt, bietet SQL die Möglichkeit, diese Änderungen zu unterbinden oder weitere Änderungen vorzunehmen, damit die Integrität wieder gewährleistet ist. Normalerweise werden im RDBMS Integritätsbedingungen nach Beendigung einer SQL-Anweisung überprüft. In bestimmten Situationen, wenn z.B. die Datenbank von einem konsistenten in einen inkonsistenten Zustand und wieder zurück überführt wird (z.B. Kontobewegungen), kann die Kontrolle der Integritätsbedingungen erst nach der Ausführung mehrerer SQL-Anweisungen sinnvoll sein. Seit SQL-92 können Integritätsbedingungen deshalb verzögert (DEFERRED) – nämlich erst nach Beendigung einer Transaktion – überprüft werden.[93]

Nach dem SQL-Standard kann zwischen zwei Arten, semantische Integritätsbedingungen zu implementieren, unterschieden werden: deklarativ oder prozedural (als sogenannte Trigger).[94]

Im Folgenden werden zunächst die wichtigsten Sprachelemente zur deklarativen Erstellung von Integritätsbedingungen beschrieben um dann die etwas komplexeren Trigger zu erläutern, über die mit den im darauffolgenden Abschnitt betrachteten prozeduralen Sprachelementen komplexere Überprüfungen möglich sind. Zu den deklarativen Integritätsbedingungen gehören:

- Datentypen
- NOT NULL
- DEFAULT
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY … REFERENCES
- DOMAIN
- CHECK
- ASSERTION

[92] Vgl. hierzu auch: Date, C.J.: An Introduction to Database Systems, Sixth Edition; Addison-Wesley; Reading, Massachusetts; 1995; S. 451
[93] Melton, J.; Simon, A.R.: SQL:1999 - Understanding Relational Language Components; Morgan Kaufmann; San Francisco; 2002; S. 360
[94] Türker, C.; Gertz, M.: Semantic integrity support in SQL:1999 and commercial (object-)relational database management systems; erschienen in: The VLDB Journal - The international Journal on Very Large Data Bases, Vol. 10, Issue 4; Springer Verlag; New York; 2001; S. 244

Die einfachste Form einer Integritätsbedingung sind Datentypen selbst, da deren Festlegung den Wertebereich eines Attributs beschränken. So lässt ein Attribut, das mit dem Datentyp INTEGER deklariert wurde, nur Ganzzahlen als Werte zu. Zusätzlich zu den vordefinierten Werten definieren alle Datentypen den NULL-Wert für einen nicht vorhandenen oder nicht bekannten Wert. Soll ein Attribut dagegen keine NULL-Werte zulassen, so wird das Attribut mit NOT NULL deklariert. Wird beim Einfügen eines Datensatzes der Wert eines Attributes nicht explizit angegeben, so wird hier der NULL-Wert als Standardwert verwendet. Über das Schlüsselwort DEFAULT kann allerdings auch ein anderer Standardwert verwendet werden, der beim Einfügen eines Datensatzes gespeichert wird, wenn der Wert des Attributes nicht explizit angegeben wird. Wird ein Attribut als UNIQUE oder PRIMARY KEY deklariert, so müssen alle Werte eindeutig sein, d.h. kein Wert darf mehr als einmal vorkommen. Im Gegensatz zu PRIMARY KEY sind bei einem als UNIQUE deklarierten Attribut auch NULL-Werte erlaubt.[95] PRIMARY KEY dient dagegen im Gegensatz zu UNIQUE der Definition eines Primärschlüsselattributes.[96]

Referentielle Integrität wird über das Sprachkonstrukt FOREIGN KEY…REFERENCES bei der Erzeugung eines Attributes erzeugt. Dabei muss das referenzierte Attribut als PRIMARY KEY oder UNIQUE deklariert sein und alle Werte des Fremdschlüssels müssen als Werte des referenzierten Attributes vorkommen. Wird der Wert des Primärschlüssels, auf den der Fremdschlüssel verweist, verändert oder gelöscht, kann über die Schlüsselwörter ON DELETE und ON UPDATE bestimmt werden, wie mit dem Wert des Fremdschlüsselattributes verfahren werden soll. Dabei sind fünf unterschiedliche Aktionen möglich: NO ACTION, SET NULL, CASCADE, SET DEFAULT und RESTRICT.

Beispiel: CREATE TABLE Abteilung (AbtNr INT PRIMARY KEY)
CREATE TABLE Personal (AbtNr INT REFERENCES Abteilung)

Eine Domain ist eine benannte, benutzerdefinierte Menge von gültigen Werten[97] und damit vergleichbar mit einem built-in Datentyp, der allerdings intrinsisch vordefinierte Wertebereiche enthält. Eine Domain wird wie ein normaler Datentyp bei der Deklaration von

[95] Garcia-Molina, H.; Ullman, J.D.; Widom, J.: Database Systems: The Complete Book; Pearson Education International; Upper Saddle River, New Jersey; 2002; S. 318
[96] Bei SQL-89 musste ein zusätzliches NOT NULL beim PRIMARY KEY mit angegeben werden. Seit SQL-92 ist dieses allerdings implizit mit enthalten.
[97] Gulutzan, P.; Pelzer, T.: SQL-99 Complete, Really; R&D Books; Lawrence, Kansas; 1999; S. 393

Attributen verwendet. Über CREATE DOMAIN wird die Domain angelegt und kann dann wie ein Datentyp genutzt werden.

Beispiel: CREATE DOMAIN SEX AS CHAR(1)
DEFAULT 'U'
CHECK(VALUE IN ('W', 'M', 'U'))

CREATE TABLE Personal (Geschlecht SEX)

Die CHECK-Anweisung dient attribut- oder tupelbasiert dem Einschränken von Wertebereichen oder Wertekombinationen über ein boolesches Prädikat.

Beispiele:

attributbasiert: CHECK (Geschlecht IN (’W’, ’M’))
tupelbasiert: CHECK(Alter<18 AND Familienstand =’ledig’)

Über die CHECK-Anweisung können auch die Integritätsbedingungen NOT NULL und UNIQUE formuliert werden:[98]

CHECK(spalte IS NOT NULL))
CHECK(UNIQUE(SELECT spalte FROM tabelle)

Über das ASSERTION-Prädikat kann eine zu erfüllende Bedingung ausdrückt werden,[99] die unabhängig von einer Tabelle deklariert wird und der Formulierung einer allgemeinen Regel entspricht. Die bisher beschriebenen deklarativen Integritätsbedingungen werden innerhalb einer Tabelle deklariert bzw. benutzt und beziehen sich damit auf diese eine Tabelle.[100] Eine ASSERTION wird deshalb normalerweise für Integritätsbedingungen über mehrere Tabellen verwendet. Ein CHECK bezieht sich immer auf eine bestimmte Tabelle. Ein ASSERTION wird dagegen immer überprüft, wenn einer der in der Bedingung vorkommenden Attribute oder Relationen sich ändert.

[98] Melton, J.; Simon, A.R.: SQL:1999 - Understanding Relational Language Components; Morgan Kaufmann; San Francisco; 2002; S. 373
[99] Silberschatz, A.; Korth, H.F.; Sudarshan, S.: Database System Concepts, Fifth Edition; McGraw-Hill; New York; 2006; S. 132
[100] Melton, J.; Simon, A.R.: SQL:1999 - Understanding Relational Language Components; Morgan Kaufmann; San Francisco; 2002; S. 373

Beispiel: CREATE ASSERTION Gehaltspruefung CHECK
((SELECT SUM(Gehalt) FROM Angestellte) +
(SELECT SUM(Gehalt) FROM Arbeiter) < 1000000)

Tabelle 1 gibt noch einmal einen Überblick über die deklarativen Integritätsbedingungen und zeigt, welche von den bekannten kommerziellen relationalen Datenbankmanagementsystemen unterstützt werden.

Tab. 1: Vergleich deklarativer Integritätsbedingungen

		SQL:1999	Oracle 8i Rel. 8.1.6	IBM DB2 Univ. V7	Informix Dyn. V9.2	MS SQL V7	Sybase Enterpr. V11.5	Ingres II Rel. 2.0	Sybase Anywh. V6.0.3
NOT NULL		√	√	√	√	√	√	√	√
DEFAULT		√	√	√	√	√	√	√	√
UNIQUE		√	√	√	√	√	√	√	√
PRIMARY KEY		√	√	√	√	√	√	√	√
FOREIGN KEY		√	(√)	(√)	(√)	(√)	(√)	(√)	(√)
ON DELETE	NO ACTION	√	(√)	√	(√)	(√)	(√)	(√)	-
	RESTRICT	√	-	√	-	-	-	-	√
	CASCADE	√	√	√	√	-	-	-	√
	SET NULL	√	√	√	-	-	-	-	√
	SET DEFAULT	√	-	-	-	-	-	-	√
ON UPDATE	NO ACTION	√	(√)	√	(√)	(√)	(√)	(√)	-
	RESTRICT	√	-	√	-	-	-	-	√
	CASCADE	√	-	-	-	-	-	-	√
	SET NULL	√	-	-	-	-	-	-	√
	SET DEFAULT	√	-	-	-	-	-	-	√
CHECK	column-level	√	√	√	√	√	√	√	√
	row-level	√	√	√	√	√	√	√	√
	table-level	√	-	-	-	-	-	-	-
	database-level	√	-	-	-	-	-	-	-
DOMAIN		√	-	-	-	-	-	-	√
ASSERTION		√	-	-	-	-	-	-	-

√ = vorhanden; (√) = teilweise vorhanden

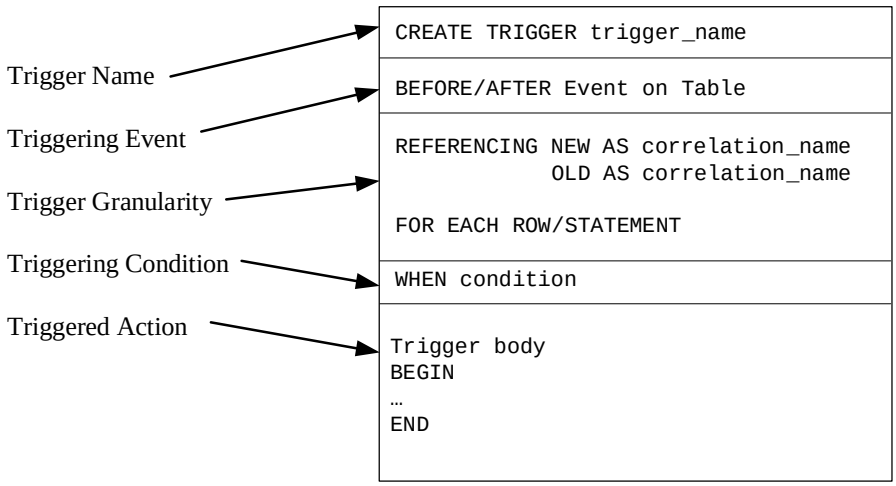
Quelle: eigene Tabelle in Anlehnung an: Türker, C.; Gertz, M.: Semantic integrity support in SQL:1999 and commercial (object-)relational database management systems; erschienen in: The VLDB Journal - The international Journal on Very Large Data Bases, Vol. 10, Issue 4; Springer Verlag; New York; 2001; S. 252

Neben den bisher beschriebenen deklarativen Möglichkeiten, konsistente widerspruchsfreie Daten sicherzustellen, existiert über sogenannte Trigger ein Sprachkonstrukt, Überprüfungen

auch prozedural vorzunehmen. Deklarativ definierte Integritätsbedingungen sind in ihrer Aufgabe wesentlich spezifischer als Trigger, da Trigger es dem Programmierer erlauben, beliebige Bedingungen und Einschränkungen zu überprüfen und selbst zu entscheiden, wie bei einer Integritätsverletzung reagiert wird.[101] Ein Trigger stellt ein aktives Sprachelement in Form einer Prozedur dar, die beim Auftreten eines Ereignisses in einer Tabelle automatisch aufgerufen wird und darauf reagiert. Trigger sind seit SQL:1999 standardisiert, obgleich sie in kommerziellen RDBMS schon vorher implementiert wurden, so dass unterschiedliche Implementierungen in der Syntax und in den Eigenschaften existieren.[102] Sie können zum einen zur Konsistenzerhaltung eingesetzt werden, aber auch zur Ereignisinformation, z.B. wenn der Lagerbestand unter einen bestimmten Wert sinkt und automatisch nachbestellt werden soll.[103]

Die folgende Abbildung zeigt den Aufbau eines Triggers:

Abb. 11: Trigger-Struktur



Quelle: Fortier, P.: SQL 3 - Implementing the SQL Foundation Standard; McGraw-Hill; New York u.a.; 1999; S. 277

Über CREATE TRIGGER wird der Triggername, das Ereignis, auf das reagiert werden soll (INSERT, UPDATE, DELETE), die Granularität (Aufruf bei jedem Datensatz oder pro Anweisung), eine allgemeine Bedingung und schließlich die Aktion festgelegt, die bei Eintreten des Ereignisses ausgeführt werden soll. Über ALTER TRIGGER können Trigger

[101] Fortier, P.: SQL 3 - Implementing the SQL Foundation Standard; McGraw-Hill; New York u.a.; 1999; S. 274
[102] Silberschatz, A.; Korth, H.F.; Sudarshan, S.: Database System Concepts, Fifth Edition; McGraw-Hill; New York; 2006; S. 330
[103] Kemper, A.; Eickler, A.: Datenbanksysteme, 6. Auflage; Oldenbourg Wissenschaftsverlag; München; 2006; S. 164

auch aktiviert oder deaktiviert werden.[104] Wird eine deklarative Integritätsbedingung verletzt, so wird ein Rollback normalerweise automatisch durchgeführt. Wird die Integrität dagegen innerhalb eines Triggers verletzt, so ist der Trigger selbst dafür verantwortlich, ob ein Rollback durchgeführt werden soll.

Aus der Idee der Trigger sind in der Forschung die aktiven Datenbankmanagementsysteme hervorgegangen, die einen allgemeinen Ansatz von Integritätsbedingungen in Form von Regeln vorsehen.

2.2.5.3 Prozedurale Sprachelemente

Teil 2 (SQL/Foundation) der SQL:2003-Standardisierung beschreibt das Erzeugen, Ändern und Löschen von Routinen in SQL. Teil 4 (SQL/PSM) beschreibt „Persistent Stored Modules (PSM)“, die 1996 im erweiterten SQL-92 Standard zum ersten Mal veröffentlicht wurden.[105] Dabei spezifiziert SQL/PSM drei verschiedene Technologien: das Verwalten von Modulen, die Routinen enthalten, prozedurale Sprachelemente und die Möglichkeit Routinen in SQL oder einer Host-Programmiersprache zu schreiben, um diese von SQL aus zu verwenden.[106] Über Routinen existiert die Möglichkeit, externen Code einer von der ISO standardisierten Programmiersprache wie C, Pascal oder Cobol einzubinden[107] oder prozedurale und deklarative SQL-Sprachelemente in den Routinen zu verwenden.[108] Dabei müssen die Routinen nicht zwangsläufig einem Modul, sondern können auch unabhängig nur einem bestimmten Schema zugeordnet werden.[109] Bei externen Routinen wird im SQL-Standard nicht festgelegt, wo (z.B. im Dateisystem oder in einer Datenbank) externe Routinen gespeichert werden.

Bei den Routinen unterscheidet man in SQL zwischen Prozeduren, die mit CREATE PROCEDURE angelegt, und Funktionen, die mit CREATE FUNCTION erzeugt werden.

---

[104] Silberschatz, A.; Korth, H.F.; Sudarshan, S.: Database System Concepts, Fifth Edition; McGraw-Hill; New York; 2006; S. 332
[105] Melton, J.; Simon, A.R.: SQL:1999 - Understanding Relational Language Components; Morgan Kaufmann; San Francisco; 2002; S. 24
[106] Melton, J.; Simon, A.R.: SQL:1999 - Understanding Relational Language Components; Morgan Kaufmann; San Francisco; 2002; S. 542
[107] Kommerzielle RDBMS unterstützen i.d.R. nicht alle im SQL standardisierten Programmiersprachen, sondern verwenden heutzutage normalerweise moderne objektorientierte Programmiersprachen wie Java oder C#
[108] Silberschatz, A.; Korth, H.F.; Sudarshan, S.: Database System Concepts, Fifth Edition; McGraw-Hill; New York; 2006; S. 145
[109] Fortier, P.: SQL 3 - Implementing the SQL Foundation Standard; McGraw-Hill; New York u.a.; 1999; S. 304f

Prozeduren werden mit CALL aufgerufen und können damit im Gegensatz zu Funktionen nicht Teil eines Werteausdrucks in einer SQL-Anweisung sein. Prozeduren haben im Gegensatz zu Funktionen keinen Rückgabewert, allerdings können sowohl Ein- als auch Ausgabeparameter angegeben werden.[110] Externe Routinen geben über die Schlüsselwörter EXTERNAL NAME einen externen Funktionsnamen an.

Beispiele:[111]

```
CREATE PROCEDURE Summe( IN z1 INTEGER, IN z2 INTEGER, OUT g INTEGER)
    SET g = z1 + z2

CREATE FUNCTION Summe (z1 INTEGER, z2 INTEGER)  RETURNS INTEGER
    RETURN z1 + z2
```

Seit SQL:2003 gibt es auch tabellenwertige Funktionen, die keinen skalaren Wert zurückliefern, sondern eine Tabelle:

CREATE FUNCTION <name>(<parameter>) RETURNS TABLE(<Spaltenliste>)

An prozeduralen Sprachelementen werden im SQL-Standard u.a. folgende Verzweigungen,

- IF…THEN…ELSEIF…ELSE…ENDIF
- CASE…WHEN…ELSE…END CASE

Schleifen,

- WHILE…DO…END WHILE
- REPEAT…UNTIL…END REPEAT
- FOR…DO…END FOR
- LOOP…END LOOP

Blöcke

- BEGIN…END

und die Behandlung von Außnahmen beschrieben.[112]

[110] Melton, J.; Simon, A.R.: SQL:1999 - Understanding Relational Language Components; Morgan Kaufmann; San Francisco; 2002; S. 545

[111] Vgl. hierzu: Türker, C.: SQL:1999 & SQL:2003 - Objektrelationales SQL, SQLJ & SQL/XML; dpunkt Verlag; 2003; S. 32f

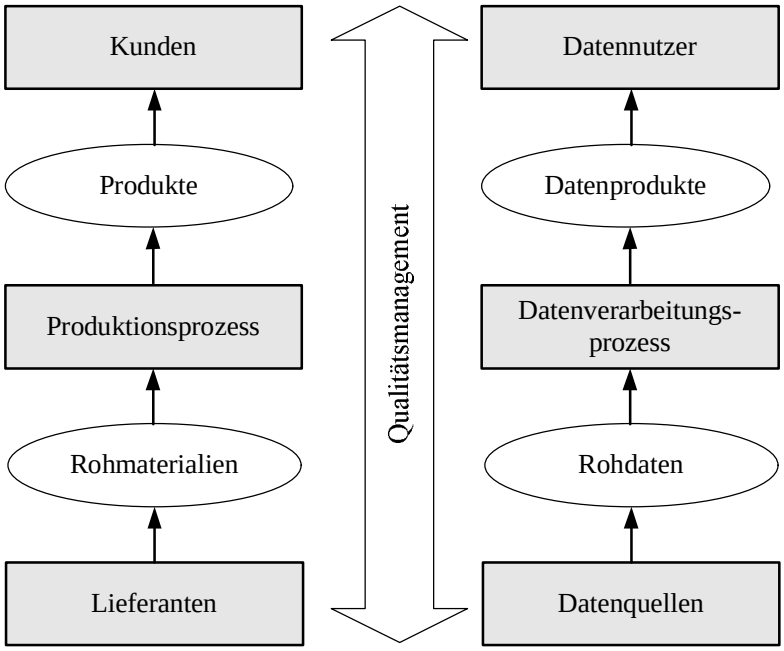
[112] Vgl. hierzu: Gulutzan, P.; Pelzer, T.: SQL-99 Complete, Really; R&D Books; Lawrence, Kansas; 1999; S. 497ff

2.3 Datenqualität

2.3.1 Einführung

Im Vergleich zu Produkten kann die Qualität einer Information nicht „besichtigt“ und vor der Verwendung überprüft werden.[113] Vielmehr kann sie erst beurteilt werden, wenn sie angewendet wird. Eine Anlehnung an den Herstellungsprozess von Produkten erscheint deshalb sinnvoll, da auch der Prozess der Verarbeitung der Daten als Herstellungsprozess betrachtet werden kann. Danach stellen die Anwender als Datennutzer die Kunden dar und die Datenquellen die Lieferanten. Hier ist durchaus eine Analogie der Anforderungen des Anwenders im Vergleich zu den Anforderungen eines Kunden an ein Produkt gegeben.

Abb. 12: Analogie zwischen industrieller Fertigung und Datenverarbeitung



Quelle: Grimmer, U.; Hinrichs, H.: Datenqualitätsmanagement mit Data-Mining-Unterstützung; erschienen in: Hildebrand, K. (Hrsg.): Business Intelligence; HMD - Praxis der Wirtschaftsinformatik 12/2001; Band 222; dpunkt Verlag; Heidelberg; 2001; S. 72

Im Folgenden werden zunächst Kategorien von Datenfehlern erörtert und deren Einteilung in Kategorien vorgestellt. Darauf aufbauend wird der Ansatz der Analogie zwischen

[113] Nohr, H.: Management der Informationsqualität, Arbeitspapiere Wissensmanagement Nr. 3/2001; Fachhochschule Stuttgart, Studiengang Informationswirtschaft; Stuttgart; 2001; S. 6

Herstellungsprozess von Produkten und Datenverarbeitungsprozess aufgegriffen und eine Definition zur Datenqualität festgelegt.

2.3.2 Datenfehlerkategorien

Datenanomalien oder Datenfehler entstehen durch eine falsche oder unzureichende Abbildung eines Ausschnitts der realen Welt in einer Datenbank. Eine unzureichende Abbildung führt bei der Verwendung des implementierten Datenmodelles zu inkorrekten Daten. H. Müller und J.-C. Freytag unterscheiden zwischen syntaktischen und semantischen Datenanomalien und Abdeckungsanomalien.[114] Zu den syntaktischen Datenanomalien zählen lexikalische Fehler, Domainformatfehler und Abweichungen. Lexikalische Fehler beziehen sich auf Diskrepanzen zwischen der definierten Struktur und der in der Verwendung benutzten Struktur. Domain-formatfehler treten auf, wenn der Wert nicht dem vorgesehenen Format entspricht. Angenommen ein Attribut Name soll die Namen einer Person in der Form „Nachname, Vorname“ speichern, so ist der Wert „Hans Muster“ inkorrekt, da zuerst der Vor- und dann der Nachname ohne Komma folgt. Abweichungen beziehen sich auf die uneinheitliche Verwendung von Werten, Maßeinheiten und Abkürzungen. Hierzu gehört z.B. die Benutzung unterschiedlicher Währungseinheiten, ohne dass diese im Attribut oder einem anderen Attribut mit angegeben werden.

Zu den semantischen Datenanomalien zählen die Verletzungen von Integritätsbedingungen, Widersprüche, Duplikate und ungültige Tupel. Verletzungen von Integritätsbedingungen beschreiben Datensätze, die bestimmte Einschränkungen nicht berücksichtigen, z.B. Alter > 0. Widersprüche sind Werte, die voneinander abhängig sind und sich widersprechen, z.B. Alter und Geburtsdatum. Duplikate sind mehrfache gespeicherte Tupel ein und desselben Objekts in der realen Welt. Unter ungültigen Tupeln versteht man Daten, die nicht den Werten der realen Welt entsprechen. Gerade solche Fehler sind schwer zu finden und üblicherweise nur durch einen Abgleich mit einem anderen Datenbestand, der korrekte Werte der realen Welt enthält, zu erkennen. Unter Abdeckungsanomalien sind schließlich fehlende Werte und fehlende Tupel zu verstehen.

[114] Müller, H.; Freytag, J.-C.: Problems, Methods, and Challenges in Comprehensive Data Cleansing, Technical Report HUB-IB-164; Humboldt Universität; Berlin; 2003; S. 6f

In Anlehnung an E. Rahm und H.H. Do können Datenfehler auch danach unterschieden werden, ob es sich um ein Problem auf Schemaebene oder Datenebene handelt.[115] Die Datenqualität als solche ist weitgehend von dem Grad abhängig, in dem Integritätsbedingungen auf Schema- und Datenebene Verwendung finden. Werden Daten in einer Datei gespeichert, so besitzen diese kein eigenes Schema und keine Einschränkungen in Bezug auf die Eingabe und Speicherung der Daten. Datenbanksysteme unterstützen zwar datenmodell- und anwendungsspezifische Einschränkungen, dennoch treten schemabezogene Datenfehler auf, weil eine Einschränkung über das Datenbanksystem nicht möglich ist oder, was der häufigere Fall ist, Einschränkungen vergessen oder einfach weggelassen werden. Datenfehler auf Datenebene entstehen durch Fehler oder Inkonsistenzen, die auf Schemaebene nicht behoben werden können, wie z.B. Rechtschreibfehler.[116] Ein allgemeines Problem hierzu, vor allem bei Legacy-Anwendungen, ist die Nutzung von Spalten für einen anderen als im Entwurf vorgesehenen Zweck.[117]

[115] Rahm, E.; Do, H.H.: Data Cleaning: Problems and Current Approaches; IEEE Bulletin of the Technical Committee on Data Engineering, Vol. 23, No. 4, December 2000; Washington, DC; S. 3-13
[116] Rahm, E.; Do, H.H.: Data Cleaning: Problems and Current Approaches; IEEE Bulletin of the Technical Committee on Data Engineering, Vol. 23, No. 4, December 2000; Washington, DC; S. 5
[117] Lee, Y.W.; Pipino, L.L.; Funk, J.D.; Wang, R.Y.: Journey to Data Quality; MIT Press; Cambridge, Massachusetts; 2006; S. 66

Abb. 13: Datenfehler bezogen auf Schema- und Datenebene

Datenfehler	
Schemaebene	**Datenebene**
• **Ungültiger Wert** Geburtsdatum = 30.13.1970 außerhalb des Wertebereichs • **Verletzung Attributabhängigkeit** Alter = 22, Geburtsdatum = 12.02.1970 • **Verletzung der Eindeutigkeit** • **Verletzung referentieller Integrität**	• **Fehlende Werte** Alter = NULL oder Alter = 999 (Dummywert) • **Rechtschreibfehler** • **Kryptische Werte, Abkürzungen** Ort = HH (für Hamburg) • **Eingebettete Werte** Name = 'Hans Muster, 12.02.1970' • **Falsche Zuordnung** Ort = Deutschland • **Widersprüchliche Werte** Ort = Hamburg, Plz = 89789 • **Worttranspositionen** Name = 'Muster, Hans', 'Hans Muster' • **Duplikate** • **Datensatzkonflikte bei Duplikaten** unterschiedliche Geburtsjahre • **Falsche Referenzwerte** Abteilungsnummer für Mitarbeiter ist vorhanden, aber nicht korrekt

Quelle: eigene Darstellung in Anlehnung an: Rahm, E.; Do, H.H.: Data Cleaning: Problems and Current Approaches; IEEE Bulletin of the Technical Committee on Data Engineering, Vol. 23, No. 4, December 2000; Washington, DC; S. 5
Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 319

2.3.3 Definition

Die Qualität eines Produktes wird nach DIN ISO 8042 als Grad der Erfüllung von Qualitätsmerkmalen definiert. Betrachtet man die Entstehung von Daten, ergibt sich zwangsläufig die Schlussfolgerung, dass Datenqualität maßgeblich von der Anwendungssoftware selbst bestimmt wird, in der die Daten produziert werden. Datenqualität ist danach vor allem auch ein grundlegendes Problem der Softwarequalität. Nach DIN ISO 9126 wird Softwarequalität analog zur Qualität eines Produktes anhand von Qualitätsmerkmalen definiert. Entsprechend sind hier u.a. folgende Softwarequalitätsmerkmale definiert, die gerade auch die Datenqualität beeinflussen:

- Funktionalität (Richtigkeit, Ordnungsmäßigkeit - Erfüllen anwendungsspezifischer Normen, Vereinbarungen, gesetzlichen Bestimmungen und ähnlichen Vorschriften),
- Benutzbarkeit,
- Effizienz (Zeitverhalten, Verbrauchsverhalten).

Nimmt man diese DIN-Normen und die Analogie zwischen Herstellungsprozess eines Produktes und Datenverarbeitungsprozess als Grundlage zur Definition von Datenqualität, so besteht der Unterschied zwischen der Qualität eines Produktes und der Qualität von Daten einzig in der Festlegung von Qualitätsmerkmalen. Genau wie die Qualitätsmerkmale eines Produktes maßgeblich vom Kunden vorgegeben werden, sind die Qualitätsmerkmale von Daten vom Anwender der Daten abhängig. A. Bauer und H. Günzel beschreiben deshalb Datenqualität als die „Eignung für einen Zweck“ (fitness for use)[118], d.h. der Grad der Datenqualität wird durch den Anwendernutzen bestimmt. Datenqualität wird also bestimmt durch die vom Anwender der Daten festgelegten Datenqualitätsmerkmale.[119]

F. Naumann vereinfacht noch stärker und definiert Datenqualität folgerichtig als einen aggregierten Wert einer Menge von Datenqualitätsmerkmalen.[120] Er weist die Auswahl der entsprechenden Datenqualitätsmerkmale Experten der Anwendungsdomäne zu.[121] Danach muss die Eignung der Daten für einen vorgesehenen Zweck für jede Anwendung individuell bestimmt werden. Betrachtet man im Vertrieb z.B. eine Mailing-Anwendung, so ist die Adresse für den Versand entscheidend. Da die Kosten beim E-Mail-Versand nicht so hoch sind wie beim postalischen, ist die Anzahl inkorrekter E-Mailadressen normalerweise nicht so entscheidend wie bei den Postadressen.[122] Bei Anwendungen wie dem Supply Chain Management (SCM) sind im Gegensatz zu Data Warehouse-Anwendungen in der Regel Datenqualitätsmerkmale wie Zeitnähe, Aktualität sowie Redundanzfreiheit wichtig.[123]

[118] Bauer, A.; Günzel H. (Hrsg.): Data Warehouse Systeme, 1. Auflage; dpunkt Verlag; Heidelberg; 2001; S. 42
[119] Helfert, M.: Proaktives Datenqualitätsmanagement in Data-Warehouse-Systemen, Dissertation; logos-Verlag; Berlin; 2002; S. 5
[120] Müller, H.; Freytag, J.-C.: Problems, Methods, and Challenges in Comprehensive Data Cleansing, Technical Report HUB-IB-164; Humboldt Universität; Berlin; 2003; S. 8
[121] Naumann, F.: Datenqualität; erschienen in: Informatik-Spektrum; Vol. 30, No. 1, Februar 2007; Springer Verlag; Heidelberg, Berlin; 2007, S. 28
[122] Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 353
[123] Hildebrandt, K.: Datenqualität im Supply Chain Management; erschienen in: Dadam, P.; Reichert, M. (Hrsg.): Informatik 2004 - Informatik verbindet, Band 1, Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V.; Bonner Köllen Verlag, Bonn, 2005; S. 239

Im Zusammenhang mit Geschäftsprozessen kann Datenqualität betrachtet werden als „Wertbemessung der Daten hinsichtlich ihrer Eignung, Prozesse abzuarbeiten“.[124]

In Anlehnung an die DIN ISO 8042 wird Datenqualität hier definiert, als der Grad der Eignung von Daten zur Erfüllung eines bestimmten Anwendungszwecks[125], der durch für diesen Anwendungszweck geeignete Qualitätsmerkmale beschrieben wird.

2.3.4 Managementansätze zur Datenqualität

2.3.4.1 TDQM (Wang)

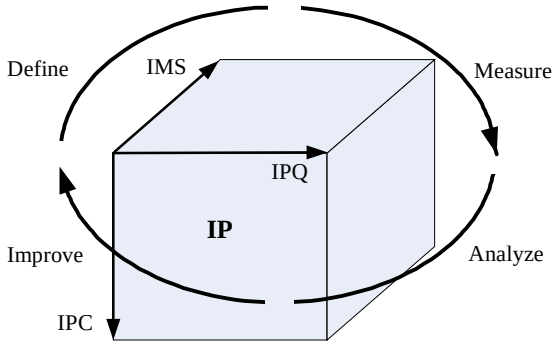
Seit 1992 wird am MIT (Massachusetts Institute of Technology) im Bereich Datenqualität geforscht und vornehmlich ein Managementansatz zur Verbesserung der Datenqualität im Rahmen des „Total Data Quality Management“ (TDQM) Programms entwickelt.[126] Beim TDQM geht es nicht so sehr um konkrete Algorithmen, sondern vielmehr um einen Managementansatz zur Veränderung des Bewusstseins der Problematik im Unternehmen.[127]

Betrachtet man, wie bereits oben erläutert, die Herstellung von Daten als Prozess, so können entsprechend Daten als Produkt angesehen werden, die innerhalb eines Prozesses erzeugt werden. „Total Data Quality Management“ (TDQM) lehnt sich deshalb an das für den Herstellungsprozess von Produkten entwickelte „Total Quality Management“ (TQM) an. In einem aus vier Phasen bestehenden Vorgehensmodell sollen qualitative Informations-Produkte (IP) erzeugt werden. Diese vier Phasen orientieren sich an dem vom TQM her bekannten Deming-Zyklus und unterteilen sich in Definieren, Messen, Analysieren und Verbessern. Ziel ist das kontinuierliche Verbessern des Informationsproduktes, indem zunächst die Merkmale der Datenqualität identifiziert und definiert werden. In der Phase des Messens werden auf Basis der identifizierten Merkmale Kennzahlen erfasst. Diese Kennzahlen werden dann in der Analysephase zur Identifikation von Problembereichen verwendet und es werden die Auswirkungen einer ungenügenden Datenqualität in Bezug auf Kosten und

[124] Würthele, V.G.: Datenqualitätsmetrik für Informationsprozesse, Diss.; Eidgenössische Technische Hochschule Zürich; Zürich; 2003; S. 23
[125] Olson, J.: Data Quality - Accuracy Dimension; Morgan Kaufman Publishers; San Francisco; 2003; S. 24
[126] Helfert, M.: Proaktives Datenqualitätsmanagement in Data-Warehouse-Systemen, Dissertation; logos-Verlag; Berlin; 2002; S. 122
[127] Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 354

Nutzen ermittelt. Schließlich werden in der Phase des Verbesserns Techniken zur Verbesserung der Datenqualität angewendet.[128]

Abb. 14: Schematische Darstellung des TDQM



Legend: IP (Information Product)
IPC (IP Characteristics)
IPQ (IP Quality)
IMS (Information Manufacturing System)

Quelle: Wang, R.Y.; Ziad, M.; Lee, Y.W.: Data Quality; Kluwer Academic Publishers; Norwell, Massachusetts; 2001; S. 6

2.3.4.2 TQdM (English)

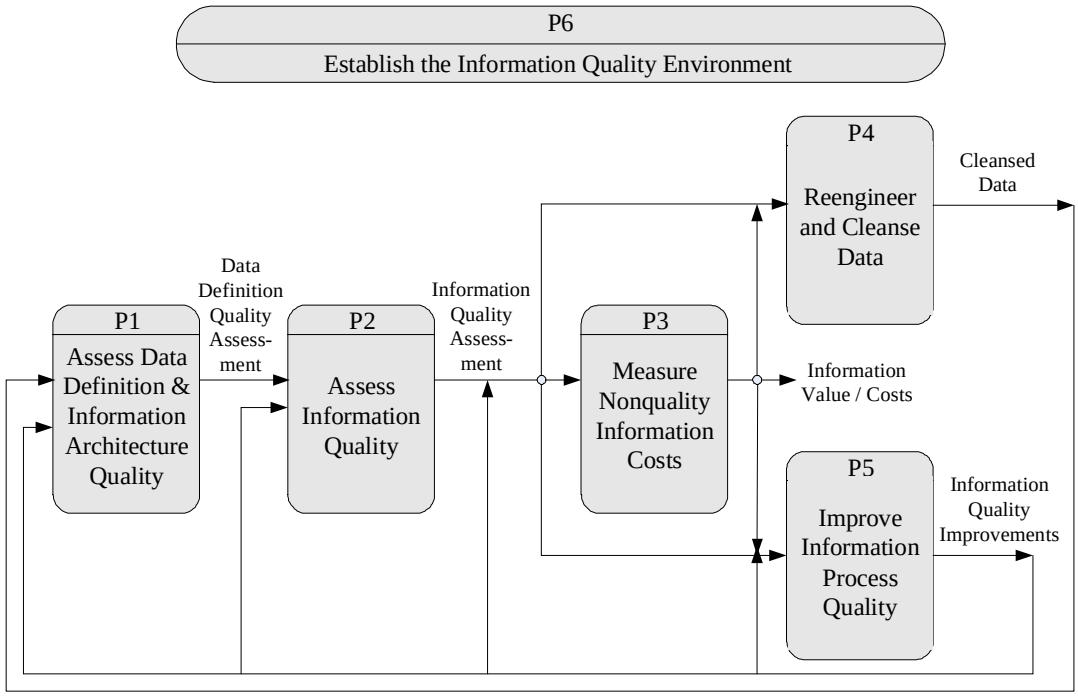
Der Managementansatz des „Total Quality data Management“ (TQdM) wurde ursprünglich von L.P. English für Data Warehouses entwickelt, ist aber eher als Methode eines allgemeinen Datenqualitätsvorgehens anzusehen. TQdM liefert viele Richtlinien, um die Kosten des Verlustes von Datenqualität und damit die Kosten von Geschäftsprozessen und der Verbesserung der Datenqualität zu bewerten. TQdM betrachtet eher den betriebswirtschaftlichen Standpunkt der Verbesserung der Datenqualität, also die Verfolgung einer Strategie, nach der sich eine Organisation richten muss, um eine hohe Datenqualität zu erreichen. Der Fokus liegt hier nicht so sehr im technischen Bereich.[129]

Das Vorgehen besteht aus fünf Schritten: Im ersten Schritt werden die wichtigsten Eigenschaften von Daten definiert. Hierzu gehören die Bezeichnungen der Datenattribute, gültige Werte und Geschäftsregeln. Als wichtigste Aufgabe in diesem ersten Schritt wird die

[128] Wang, R.Y.; Ziad, M.; Lee, Y.W.: Data Quality; Kluwer Academic Publishers; Norwell, Massachusetts; 2001; S. 5ff
[129] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 174f

Zufriedenheit der Benutzer mit der Definition der Daten ermittelt. Auf diese Basis kann dann eine Messung der Datenqualität im zweiten Schritt erfolgen. Im dritten Schritt werden die Kosten ungenügender Datenqualität gemessen und im vierten die Daten überarbeitet und bereinigt. In Schritt 5 schließlich werden die grundlegenden Probleme unzureichender Datenqualität in Geschäftsprozessen ermittelt, um die Geschäftsprozesse selbst zu verbessern.[130]

Abb. 15: TQdM Methoden Übersicht



Quelle: English, L.P.: Improving Data Warehouse and Business Information Quality; Wiley Computer Publishing; New York; 1999; S. 70

2.3.5 Datenqualitätsmerkmale

2.3.5.1 Datenqualitätsmerkmale und Metriken

Damit Datenqualität über Kennzahlen bewertet werden kann, müssen die Datenqualitätsmerkmale gemessen werden. Es sind also geeignete Merkmale auszuwählen und zu operationa-

[130] English, L.P.: Improving Data Warehouse and Business Information Quality; Wiley Computer Publishing; New York;1999; S. 69ff

lisieren. Es ist notwendig, Datenqualität in Form von konkreten numerischen Metriken zu messen, um entscheiden zu können, welche Daten mangelnde Qualität aufweisen.[131]

Im Folgenden wird exemplarisch auf die wichtigsten Qualitätsmerkmale eingegangen. Eine allgemein anerkannte Definition dieser Qualitätsmerkmale und der entsprechenden Kennzahl existiert zurzeit nicht, da viele Merkmale subjektiv zu bewerten sind. Es ist jedoch eine Liste von Merkmalen sinnvoll, um die für die Anwendung relevanten Dimensionen auszuwählen.[132]

Da Datenqualität teilweise intuitiv ist und Daten eben keine physischen Eigenschaften wie Gewicht oder Größe haben[133], ist eine ausschließlich quantitative Messung nicht sinnvoll.[134] Hinzu kommt, dass die Bedeutung einzelner Datenqualitätsmerkmale und die Messung dieser sich in der Praxis z.B. durch neue gesetzliche Regelungen und unternehmensinterne Anforderungen über die Zeit ändert.[135]

Das Unternehmen muss entscheiden, welche Datenqualitätsmerkmale wichtig sind und wie diese gemessen werden sollen, da viele Merkmale normalerweise multivariat sind. Eine allgemeine quantitative Definition der Metrik eines Datenqualitätsmerkmals sieht wie folgt aus:[136]

Bewertungskennzahl = 1 – (Anzahl unerwünschter Ergebnisse / Anzahl aller Ergebnisse)

Im Folgenden werden exemplarisch die wichtigsten Datenqualitätsmerkmale und deren Messung betrachtet.[137]

---

[131] Naumann, F.: Datenqualität; erschienen in: Informatik-Spektrum; Vol. 30, No. 1, Februar 2007; Springer Verlag; Heidelberg, Berlin; 2007, S. 28
[132] Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 354
[133] Redman, T.: Data Quality for the Information Age; Artech House; Norwood; 1996; S.186
[134] Caspers, P.; Gebauer, M.: Reproduzierbare Messung von Datenqualität mit Hilfe des DQ-Messtools WestLB-DIME; erschienen in: Dadam, P.; Reichert, M. (Hrsg.): Informatik 2004 - Informatik verbindet, Band 1, Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V.; Bonner Köllen Verlag, Bonn, 2005; S. 234
[135] Caspers, P.; Gebauer, M.: Reproduzierbare Messung von Datenqualität mit Hilfe des DQ-Messtools WestLB-DIME; erschienen in: Dadam, P.; Reichert, M. (Hrsg.): Informatik 2004 - Informatik verbindet, Band 1, Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V.; Bonner Köllen Verlag, Bonn, 2005; S. 234
[136] Lee, Y.W.; Pipino, L.L.; Funk, J.D.; Wang, R.Y.: Journey to Data Quality; MIT Press; Cambridge, Massachusetts; 2006; S. 54f
[137] Vgl. hierzu: Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 20ff
Lee, Y.W.; Pipino, L.L.; Funk, J.D.; Wang, R.Y.: Journey to Data Quality; MIT Press; Cambridge, Massachusetts; 2006; S. 55ff

Fehlerfreiheit/Korrektheit/ Genauigkeit bestimmt, in wie weit die Daten korrekt sind, d.h. zu welchem Grad zwei Werte v und v' übereinstimmen. Dabei entspricht v' dem Wert in der Realwelt. Ist v'=Hans und v=Hns, so ist v zu einem gewissen Grade inkorrekt. Bei Korrektheit kann man unterscheiden zwischen syntaktischer und semantischer Korrektheit. Bei syntaktischer Korrektheit kommt es nicht darauf an, ob ein Wert v mit dem wahren Wert v' übereinstimmt. Angenommen, es wurde ein Bereich bzw. eine Domain D gültiger Werte für ein Objekt angegeben, so ist bei syntaktischer Korrektheit entscheidend, ob der Wert v einem der Werte in D entspricht. Angenommen v enthält den Wert „Harald", so ist dieser syntaktisch korrekt, wenn er einem der Werte der Domain entspricht. Er ist aber nicht semantisch korrekt, da er nicht dem Wert in der Realwelt entspricht. Der Grad syntaktischer Korrektheit kann über Codes, Proximitätsmaße oder Nachschlagetabellen überprüft werden (siehe Abschnitt 3.2). Semantische Korrektheit dagegen kann z.B. über Nachschlagetabellen kontrolliert werden, deren Werte als semantisch korrekt bezogen auf die Realwelt gelten. Daraus ergibt sich, dass syntaktische Korrektheit normalerweise einfacher zu ermitteln ist als semantische, die voraussetzt, dass man den realen Wert kennt oder weitgehend deduktiv ermitteln kann. Typografische Fehler wie z.B. v=Hans und v'=Hns zeigen, dass syntaktische und semantische Korrektheit durchaus zusammenfallen können. Dies ist der Fall wenn Werte eng beieinander liegen, so dass man inkorrekte Werte mit möglichst ähnlichen in der Domain enthaltenen Werten korrigieren kann.[138]

Eine Metrik hierzu könnte wie folgt definiert sein:

Fehlerfreiheit = 1 – (Anzahl Fehler in Dateneinheiten / Gesamtanzahl Dateneinheiten)

Bei dieser Metrik ist festzulegen, wie die Granularität einer Dateneinheit spezifiziert ist (Datenbank, Tabelle, Spalte) und woraus sich ein Fehler ergibt.[139] Korrektheit kann sich entweder auf eine Spalte, eine Tabelle oder auch auf eine ganze Datenbank beziehen. So kann z.B. Tabellen- oder Datenbank-Korrektheit ermittelt werden, indem das Verhältnis zwischen den korrekten Werten und allen Werten errechnet wird. Im Falle einer vollständigen

[138] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 20f

[139] Scannapieco, M.; Missier, P.; Batini, C.: Data Quality at a Glance; erschienen in: Datenbank-Spektrum, 5. Jahrgang, Heft 14; dpunkt Verlag; Heidelberg; 2005; S. 6

Tabelle ermittelt man alle korrekten Zellwerte und bildet das Verhältnis zu der Anzahl aller Zellen der Tabelle.[140]

Vollständigkeit ist der Quotient aus fehlenden Werten in den Datensätzen und der Gesamtanzahl von Werten. H. Müller und H.-C. Freytag bezeichnen diese Dimension auch als Dichte.[141] Hierbei kann unterschieden werden zwischen

- Schemavollständigkeit, inwieweit evtl. Tabellen oder Attribute fehlen
- Spaltenvollständigkeit, inwieweit Werte in einer Spalte fehlen und
- Vollständigkeit der Grundgesamtheit, inwieweit alle Entitäten einer Grundgesamtheit gespeichert wurden (z.B. in einer Tabelle Bundesland werden alle Bundesländer in Deutschland gespeichert oder eine Tabelle Kunden soll sich auf die Grundgesamtheit aller Einwohner Deutschlands beziehen)

Vollständigkeit kann wie folgt gemessen werden:

Vollständigkeit = 1 – (Anzahl unvollständiger Einheiten / Gesamtanzahl Einheiten)

Wird in relationalen Datenbanken NULL zur Berechnung der Vollständigkeit verwendet, so muss berücksichtigt werden, dass NULL drei verschiedene Bedeutungen haben kann:

1) Ein Wert existiert nicht
2) Ein Wert ist vorhanden, aber unbekannt
3) Es ist nicht bekannt, ob ein Wert existiert

In der ersten Bedeutung würde NULL nicht für Unvollständigkeit, in der zweiten dafür stehen und in der dritten Bedeutung wäre beides möglich.[142]

Abhängig von der Granularität der Vollständigkeit kann unterschieden werden zwischen

[140] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 22
[141] Müller, H.; Freytag, J.-C.: Problems, Methods, and Challenges in Comprehensive Data Cleansing, Technical Report HUB-IB-164; Humboldt Universität; Berlin; 2003; S. 9
[142] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 24

Datensatz-, Spalten- und Tabellen-Vollständigkeit.[143] Datensatz-Vollständigkeit bezieht sich auf die Anzahl der NULL-Werte aller Spalten eines Datensatzes, Spalten-Vollständigkeit auf die einer einzelnen Spalte im Verhältnis zu allen Werten dieser Spalte. Tabellen-Vollständigkeit ermittelt die Anzahl aller NULL-Werte einer Tabelle und setzt diese ins Verhältnis zu allen Werten der Tabelle.

Tab. 2: Beispiel für Datensatz-, Spalten- und Tabellen-Vollständigkeit

Nr	**Name**	**Vorname**	**Alter**	**Tupel-Vollständigkeit**
1	Muster	Hans	51	1,00
2	Muster	NULL	54	0,75
3	Klein	Friedhelm	NULL	0,75
4	Klein	NULL	NULL	0,50
5	Müller	Elfriede	88	1,00
6	Müller	Gutfried	NULL	0,75
Spalten-Vollständigkeit	1,00	0,67	0,50	**0,79 1-(5/24)**

Quelle: eigene Tabelle in Anlehnung an: Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 27

Konsistenz (Widerspruchsfreiheit) kann aus verschieden Perspektiven betrachtet werden. Referentielle Integrität ist dabei eine Form von Konsistenz. Allgemein betrachtet kann Konsistenz als die Widerspruchsfreiheit zwischen zwei oder mehreren zusammengehörigen Datenelementen[144] (Postleitzahl und Ort) beschrieben werden. Daneben ist aber auch die Konsistenz im Format für das gleiche Datenelement wichtig.

Konsistenz kann wie folgt gemessen werden:

Konsistenz =

1 – (Anzahl inkonsistenter Einheiten / Anzahl durchgeführter Konsistenzprüfungen)

[143] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 25f

[144] Vgl.hierzu: Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 30

Zeitnähe bezieht sich darauf, wie schnell eine Änderung an einem Datum vorgenommen wird.

Flüchtigkeit gibt an, wie oft ein Datum sich in einer vorgegebenen Zeit ändert. Stabile Daten wie Geburtstag haben normalerweise kaum Flüchtigkeit, also einen Wert nahe 0. Dynamische Daten dagegen, wie Kontobewegungen weisen eine hohe Flüchtigkeit auf.[145]

Aktualität gibt an, wie aktuell Daten für eine bestimmte Aufgabe sind. Dies ist bei Daten entscheidend, die bei nicht rechtzeitiger Änderung nutzlos für eine bestimmte Aufgabe sein können[146] (z.B. Börsenkurse).

Interpretierbarkeit ist das Maß der korrekten Interpretation der Bedeutung und der Eigenschaften von Datenquellen anhand der vorhandenen Dokumentation und Metadaten.[147]

R.Y. Wang und D.M. Strong schlagen Merkmale zur Bewertung einer Datenquelle als Ganzes vor. Hierzu gehören

- Glaubwürdigkeit (Grad, inwieweit Daten als wahr bzw. glaubhaft in Bezug zur Realwelt betrachtet werden können),
- Reputation (Grad, inwieweit eine Datenquelle als vertrauenswürdig angesehen werden kann) und
- Objektivität (Grad der Unbefangenheit bei der Datenbeschaffung).[148]

Der Grad der Glaubwürdigkeit ist hierbei abhängig von den Erfahrungen und dem Wissen der Anwender. Die Glaubwürdigkeit ist subjektiv umso höher, je eher sie mit Erfahrungen des Anwenders bezogen auf Verteilung der Daten, Datenmuster oder Wertebereiche übereinstimmen.[149] Letztendlich müssen die Erfahrungen des Anwenders mit in das Informations-

[145] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 29
[146] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 29
[147] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 33
[148] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 35
[149] Helfert, M.: Proaktives Datenqualitätsmanagement in Data-Warehouse-Systemen, Dissertation; logos-Verlag; Berlin; 2002; S. 148

system einfließen, das die Daten in einer Datenbank speichert, da diese Erfahrungen die Geschäftsregeln, Konsistenzbedingungen und Einschränkungen widerspiegeln.

Aus dem Gebiet der Informationsintegration werden zudem noch weitere relevante Merkmale wie Antwortzeit, Latenzzeit und Preis der Daten erwähnt.[150]

Zum Schluss sei noch einmal erwähnt, dass eine Festlegung von Datenqualitätsmerkmalen und deren Einteilung in Form einer hierarchischen Abbildung immer abhängig sein sollte vom Anwendungsfall. Eine quasi dogmatische Anwendung eines der im nächsten Abschnitt beschriebenen Klassifikationsansätze ist nicht sinnvoll. Es sollte immer eine Auswahl von Merkmalen aus Sicht der Anwendung und des Unternehmens erfolgen.

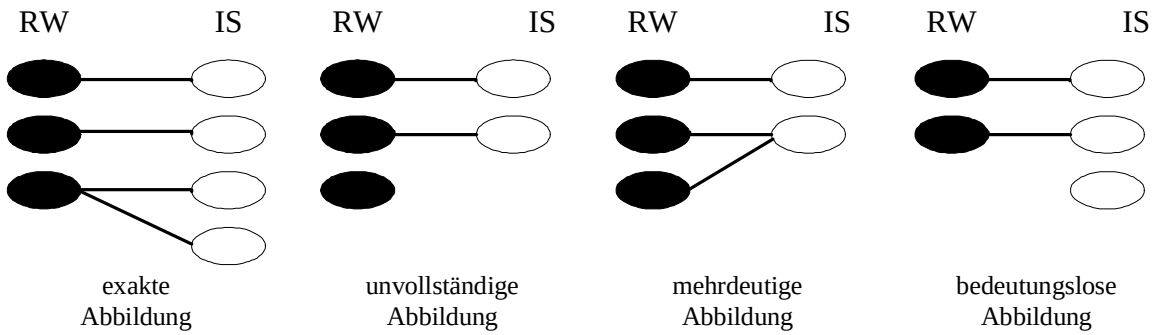
2.3.5.2 Klassifikationsansätze

In der Forschung gibt es viele Ansätze zur Einteilung von Qualitätseigenschaften in Kategorien, von denen im Folgenden die fünf bekanntesten Klassifikationsansätze erläutert werden.

Beim ersten Ansatz nach Y. Wand und R.Y. Wang handelt es sich um einen ausschließlich theoretischen Ansatz.[151] Wie bereits in der Einführung erwähnt, ist ein Informationssystem (IS) die Abbildung eines Ausschnitts der Realwelt (RW). Die Realwelt ist dann korrekt umgesetzt, wenn die Realwelt genau in das Informationssystem abgebildet wurde (RW **à** IS) und zwei oder mehrere Zustände der Realwelt nicht in einem Zustand im Informationssystem abgebildet werden.[152]

[150] Naumann, F.: Datenqualität; erschienen in: Informatik-Spektrum; Vol. 30, No. 1, Februar 2007; Springer Verlag; Heidelberg, Berlin; 2007, S. 28
[151] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 36ff
[152] Wand, Y.; Wang, R.Y.: Anchoring Data Quality Dimensions in Ontological Foundations; erschienen in: Communications of the ACM, Vol. 39, No. 11 (November 1996); New York; 1996; S. 91

Abb. 16: Abbildungen Realwelt zu Informationswelt nach Wand/Wang



Quelle: eigene Darstellung in Anlehung an: Wand, Y.; Wang, R.Y.: Anchoring Data Quality Dimensions in Ontological Foundations; erschienen in: Communications of the ACM, Vol. 39, No. 11 (November 1996); New York; 1996; S. 90

Unzulänglichkeiten in der Abbildung von der RW in das IS werden dabei unterschieden in Design-Mängel und Ausführungs-Mängel. Design-Mängel werden gemäß Abb. 16 wiederum in unvollständige, mehrdeutige und bedeutungslose Abbildungen unterteilt. Unter Ausführungs-Mängeln versteht man das sogenannte „Garbling“ („Entstellen“). Damit ist die falsche Zuordnung eines Zustands der Realwelt in die Informationswelt gemeint. „Garbling“ entsteht danach durch inkorrekte Benutzeraktionen, z.B. fehlerhafte Dateneingaben.[153] Auf Grundlage der beschriebenen Mängel werden nun Datenqualitätseigenschaften abgeleitet. Die Dimension Inkorrektheit beinhaltet danach z.B., dass das Informationssystem einen Zustand der Realwelt unterschiedlich darstellt.[154] Da es sich bei diesem Ansatz um einen rein theoretischen Ansatz handelt, werden die funktionalen Anforderungen der Endbenutzer an das Informationssystem nicht berücksichtigt.[155]

Beim empirischen Ansatz von R.Y. Wang und D.M. Strong wurden 15 verschiedene Dimensionen aufgrund von Interviews mit Datennutzern ausgewählt. Diese wurden wiederum in vier Kategorien eingeteilt.[156] Intrinsische Datenqualität ist danach die Qualität, die Daten inhärent in sich enthalten wie z.B. Korrektheit. Kontextabhängige Datenqualität ist abhängig vom Kontext, in dem die Daten genutzt werden. Die Dimension Vollständigkeit ist z.B. abhängig von der Aufgabe, die zu erledigen ist und inwieweit hierfür Daten vollständig oder

[153] Wand, Y.; Wang, R.Y.: Anchoring Data Quality Dimensions in Ontological Foundations; erschienen in: Communications of the ACM, Vol. 39, No. 11 (November 1996); New York; 1996; S. 92f
[154] Wand, Y.; Wang, R.Y.: Anchoring Data Quality Dimensions in Ontological Foundations; erschienen in: Communications of the ACM, Vol. 39, No. 11 (November 1996); New York; 1996; S. 93
[155] Winter, M.; Helfert, M.; Herrmann, C.: Das metadatenbasierte Datenqualitätssystem der Credit Suisse; erschienen in: von Maur, E.; Winter, R. (Hrsg.): Vom Data Warehouse zum Corporate Knowledge Center - Proceedings der Data Warehousing 2002; Physica-Verlag; Heidelberg; 2002; S. 162
[156] Helfert, M.: Proaktives Datenqualitätsmanagement in Data-Warehouse-Systemen, Dissertation; logos-Verlag; Berlin; 2002; S. 78

nur zum Teil vorhanden sein müssen. Darstellungsqualität bezieht sich auf Aspekte der Repräsentation. Hierzu gehört z.B. die Interpretierbarkeit oder die Konsistenz in der Präsentation der Daten (z.B. Datenformat). Datenqualität bezüglich der Zugänglichkeit der Daten bezieht sich auf das Vorhandensein und die Möglichkeiten des sicheren Zugriffs auf die Daten.[157]

Tab. 3: Klassifikationsansatz nach Wang/Strong

Category	**Dimension**	**Definition: the extent to which…**
Intrinsic	Believability	data are accepted or regarded as true, real and credible
	Accuracy	data are corrected, reliable and certified free of error
	Objectivity	data are unbiased and impartial
	Reputation	data are trusted or highly regarded in terms of their source and content
Contextual	Value-added	data are beneficial and provide advantages for their use
	Relevancy	data are applicable and useful for the task at hand
	Timeliness	the age of the data is appropriate for the task at hand
	Completeness	data are of sufficient depth, breadth, and scope for the task at hand
	Appropriate amount of data	the quantity or volume of available data is appropriate
Representational	Interpretability	data are in appropriate language and unit and the data definitions are clear
	Ease of understanding	data are clear without ambiguity and easily comprehended
	Representational consistency	data are always presented in the same format and are compatible with the previous data
	Concise representation	data are compactly represented without being overwhelmed
Accessibility	Accessibility	data are available or easily and quickly retrieved
	Access security	access to data can be restricted and hence kept secure

Quelle: Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 38

Beim intuitiven Ansatz nach T.C. Redman werden drei Kategorien von Dimensionen unterschieden: Dimensionen zum konzeptionellen Schema, zu Datenwerten und zum Datenformat.[158] Dimensionen des konzeptionellen Schemas fassen solche Merkmale zusammen, die sich auf das Datenschema, also auf das Modell der Abbildung der realen Welt, beziehen. Dimensionen zu Datenwerten beziehen sich auf den Inhalt ohne Berücksichtigung

[157] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 39
[158] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 39

der Darstellung der Daten. Dimensionen zum Datenformat fassen Merkmale zur Darstellung der Daten zusammen.[159]

Tab. 4: Klassifikationsansatz nach Redman

The Conceptual View			
Content	relevance	obtainability	clarity of definition
Scope	comprehensiveness	essentialness	
Level of Detail	attribute granularity	precision of domains	
Composition	naturalness	identifiability	
	homogeneity	minimum unnecessary redundancy	
View Consistency	semantic consistency	structural consistency	
Reaction to Change	robustness	flexibility	
Values			
	accuracy	Completeness (entities and attributes)	
	consistency	currency/cycle time	
Representation			
Formats	appropriateness	format precision	efficient of storage
	interpretability	format flexibility	
	portability	ability to represent null values	
Physical Instances	representation consistency		

Quelle: Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 38

M. Helfert stellt einen empirischen Ansatz vor, der zwar auch für Data Warehouse Systeme konzipiert wurde, aber durchaus allgemeingültig ist. M. Helfert unterscheidet in seinem Ansatz zwischen Designqualität und Ausführungsqualität. Zunächst liegt der Fokus bei der Entwicklung eines Informationssystems in der Spezifikation der Anforderungen, also im Design eines Datenbankmodells.[160] Die Designqualität bestimmt, wie gut das Informationssystem den Ausschnitt der realen Welt abbildet. Ist das Informationssystem im Einsatz, ändert sich die Zielsetzung auf die Einhaltung der im Design spezifizierten Anforderungen, der Ausführungsqualität. Abhängig davon werden Qualitätsmerkmale bezogen auf das Datenschema (Designqualität) und auf Datenwerte unterschieden.[161]

[159] Redman, T.: Data Quality for the Information Age; Artech House; Norwood; 1996; S. 245ff
[160] Helfert, M.: Proaktives Datenqualitätsmanagement in Data-Warehouse-Systemen, Dissertation; logos-Verlag; Berlin; 2002; S. 68
[161] Helfert, M.: Proaktives Datenqualitätsmanagement in Data-Warehouse-Systemen, Dissertation; logos-Verlag; Berlin; 2002; S. 81ff

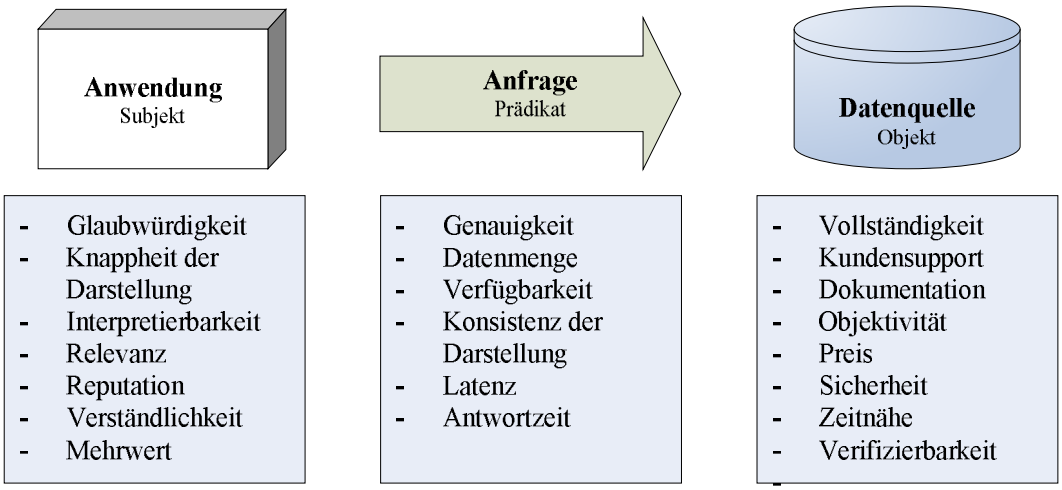
Tab. 5: Klassifikationsansatz nach Helfert

Qualitätsmerkmale bezogen auf das Datenschema		
Kategorie	**Merkmal**	**Beschreibung**
Interpretierbarkeit	Semantik	Die Entitäten, Beziehungen und Attribute und deren Wertebereiche sind einheitlich, klar und genau beschrieben sowie dokumentiert.
	Identifizierbarkeit	Einzelne Informationsobjekte (z.B. Kunden) können eindeutig identifiziert werden.
	Synonyme	Beziehungen zwischen Synonymen sind bekannt und dokumentiert.
	Zeitlicher Bezug	Der zeitliche Bezug einzelner Informationsobjekte ist abgebildet.
	Repräsentation fehlender Werte	Fehlende Werte (Nullwerte/Default-Werte) sind definiert und können abgebildet werden.
Nützlichkeit (Zweckbezogen)	Vollständigkeit	Alle wesentlichen Entitäten, Beziehungen und Attribute sind erfasst. Die Daten ermöglichen die Erfüllung der Aufgabe.
	Erforderlichkeit	Definition von Pflicht- und Kannfeldern
	Granularität	Die Entitäten, Beziehungen und Attribute sind im notwendigen Detaillierungsgrad erfasst.
	Präzision der Wertebereichsdefinition	Die Definition der Wertebereiche repräsentiert die möglichen und sinnvollen Datenwerte.
Qualitätsmerkmale bezogen auf die Datenwerte		
Glaubwürdigkeit	Korrektheit	Die Daten stimmen inhaltlich mit der Datendefinition überein und sind empirisch korrekt.
	Datenherkunft	Die Datenherkunft und die vorgenommenen Datentransformationen sind bekannt.
	Vollständigkeit	Alle Daten sind gemäß Datenmodell erfasst.
	Widerspruchsfreiheit	Die Daten weisen keine Widersprüche zu Integritätsbedingungen (Geschäftsregeln, Erfahrungswerte) und Wertebereichsdefinitionen auf (innerhalb des Datenbestandes, zu anderen Datenbeständen, im Zeitverlauf)
	Syntaktische Korrektheit	Die Daten stimmen mit der spezifizierten Syntax (Format) überein.
	Zuverlässigkeit	Die Glaubwürdigkeit der Daten ist konstant
Zeitlicher Bezug	Aktualität	Datenwerte bezogen auf den gegenwärtigen Zeitpunkt sind erfasst.
	Zeitliche Konsistenz	Alle Datenwerte bzgl. eines Zeitpunktes sind gleichermassen aktuell.
	Nicht-Volatilität	Die Datenwerte sind permanent und können zu einem späteren Zeitpunt wieder aufgerufen werden.
Nützlichkeit	Relevanz	Die Datenwerte können auf einen relevanten Datenausschnitt beschränkt werden.
	Zeitlicher Bezug	Die Datenwerte beziehen sich auf den benötigten Zeitraum.
Verfügbarkeit	Zeitliche Verfügbarkeit	Die Datenwerte stehen rechtzeitig zur Verfügung.
	Systemverfügbarkeit	Das Gesamtsystem ist verfügbar.
	Transaktionsverfügbarkeit	Einzelne benötigte Transaktionen sind ausführbar, die Zugriffszeit ist akzeptabel und gleichbleibend.
	Zugriffsrechte	Die benötigten Zugriffsrechte sind ausreichend.

Quelle: Helfert, M.: Proaktives Datenqualitätsmanagement in Data-Warehouse-Systemen, Dissertation;

U. Leser und F. Naumann klassifizieren die Qualitätsmerkmale in Abhängigkeit des Gesamtprozesses der Anfragebearbeitung. Dabei wird unterschieden zwischen der Anfrageerzeugung durch den Benutzer, der Anfrageverarbeitung und der Datenquelle selbst. Diesen drei als Subjekt, Prädikat und Objekt bezeichneten Klassen werden die Dimensionen wie in Abb. 17 dargestellt zugeordnet:[162]

Abb. 17: Klassifikationsansatz nach Leser/Naumann



Quelle: : Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 357

Unter die Klasse Subjekt fallen Kriterien, die der Anwender selbst bewertet (z.B. Expertenbefragung, Fragebögen). Zur Klasse Prädikat gehören dagegen weitgehend objektiv messbare Dimensionen, die während der Anfragebearbeitung gemessen und bewertet werden können. Die Klasse Objekt umfasst schließlich Dimensionen, wie z.B. Vollständigkeit oder Dokumentation, die von der Datenquelle selbst oder durch einen Experten beantwortet werden können.[163] Dieser Klassifikationsansatz veranschaulicht vor allem das Problem objektiver und subjektiver Kennzahlen. Kriterien der Klasse Subjekt sind nicht quantitativ messbar und müssen normalerweise durch den Anwender erfragt werden.[164]

Generell gibt es keinen einheitlichen Klassifikationsansatz, der vorgibt, welche Dimensionen die Datenqualität bestimmen, in welche Kategorien sie unterteilt werden und wie sie

[162] Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 356f
[163] Naumann, F.; Rolker, C.: Assessment Methods for Information Quality Criteria, Informatik-Bericht Nr. 138; Humboldt-Universität zu Berlin; Berlin; 2000; S. 6ff
[164] Vgl. hierzu: Naumann, F.; Leser, U.; Freytag, J. C.: Quality-driven Integration of Heterogeneous Information Systems, Informatik-Bericht Nr. 117; Humboldt-Universität zu Berlin; Berlin; 1999; S. 8

gemessen werden.[165] Einer der Hauptgründe ist darin zu sehen, dass Daten unterschiedliche Bereiche der realen Welt abbilden sollen und Daten normalerweise domänenabhängig sind[166]. Dadurch kommen natürlich auch sehr spezifische Dimensionen zum Tragen. Die hier vorgestellten Datenqualitätsmerkmale könnten z.B. noch weiter aufgeschlüsselt werden. Genauigkeit könnte z.B., falls dies erforderlich ist, unterschieden werden in syntaktische und semantische Genauigkeit.

[165] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 39

[166] Ein Beispiel für die Domänenabhängigkeit von Datenqualität in software-technischer Hinsicht findet sich in: Rusch, B.: Normierung von Zeichenfolgen als erster Schritt des Match; Preprint SC-99-13; Konrad-Zuse-Zentrum für Informationstechnik Berlin; Berlin; 1999

3. Aufgaben einer Datenqualitätskomponente

3.1 Problematik

Die OMG (Object Management Group) definiert ein Geschäftsobjekt (business object) als "a representation of a thing active in the business domain, including at least its business name and definition, attributes, behaviour, relationships, rules, policies, and constraints."[167] Betrachtet man also Datenprobleme, so hat dieses immer mit einem Verständnis von Regeln und dem Verhalten von Geschäftsobjekten zu tun. Um Probleme mit Daten bzw. Geschäftsobjektdaten zu bereinigen, müssen diese zunächst analysiert und erkannt werden. Das Analysieren und Erkennen von Datenproblemen setzt ein detailliertes Domänenwissen voraus, so dass eine Verbesserung der Datenqualität nur in Zusammenarbeit mit Domänenexperten durchgeführt werden sollte, die die Eigenschaften von Geschäftsobjekten bewerten können. Das wiederum schließt aber auch ein weitgehend automatisiertes Bereinigen der Daten aus, so dass Datenbereinigung selten automatisch, eher semi-automatisch erfolgen kann.[168]

Generell lassen sich drei wichtige Teilbereiche in einer Datenqualitätskomponente unterscheiden:

1) Verstehen
2) Verbessern
3) Steuern

Zum Verstehen gehört die Analyse des Dateninhaltes und der Datenstrukturen. Zum Verbessern der Datenqualität zählt das Vereinheitlichen in ein konsistentes Datenformat, das Anreichern von Daten und die Duplikaterkennung. Zum Steuern gehört schließlich das Messen und kontinuierliche Überwachen von Datenqualitätsmerkmalen.

Im Folgenden werden zunächst allgemeine Grundlagen und Algorithmen erläutert, die in den drei Teilkomponenten eingesetzt werden. Danach werden die drei Teilkomponenten auf Basis dieser Grundlagen detailliert beschrieben.

[167] Object Management Group Common Facilities RFP-4: Common business objects and business object facility; OMG TC Document CF/96-01-04; 1996; Framingham; S. 19
[168] Müller, H.; Freytag, J.-C.: Problems, Methods, and Challenges in Comprehensive Data Cleansing, Technical Report HUB-IB-164; Humboldt Universität; Berlin; 2003; S. 3

3.2 Grundlagen

3.2.1 Codes

3.2.1.1 Grundlagen

In der Codierungstheorie versteht man unter einem Code die Abbildung von einem Quellalphabet in ein Zielalphabet. Dabei ist die Abbildung injektiv, zwei unterschiedliche Quellzeichen dürfen also nicht auf das gleiche Zielzeichen abgebildet werden. In der Codierungstheorie geht es dabei um die Fehlererkennung bei der Übertragung von Daten. Hierzu werden den Nutzdaten zusätzliche redundante Daten hinzugefügt, um Fehler bei der Übertragung zu erkennen.[169] Ein typisches Beispiel hierfür sind z.B. die Prüfzifferncodes, die in 3.2.1.4 beschrieben werden.

Bei der Erkennung von Datenqualitätsproblemen geht es neben präventiven Maßnahmen zur Fehlerverhinderung, wie z.B. über Prüfzifferncodes, auch um das Erkennen von Unschärfen in den Daten. Im Rahmen dieser Arbeit soll der Begriff der Codierung deshalb weitergefasst werden und darunter auch die nicht umkehrbare Abbildung von einem Quellalphabet in ein Zielalphabet verstanden werden, um so Unschärfen in den Daten zu erkennen. Ein einfaches Beispiel hierfür ist z.B. die Abbildung aller Ganzzahlen durch die Zahl ‚9' und aller alphabetischen Zeichen durch den Buchstaben ‚A'. In einem Attribut Alter könnte man dann über eine Häufigkeitsverteilung die Datensätze finden, in denen in diesem Attribut ein nicht erlaubtes Zeichen eingegeben wurde.[170]

Die funktionale Abbildung von Nutzdaten in einen Code kann also sowohl umkehrbar (mit redundanten Zusatzdaten zur Fehlererkennung[171]) und nicht umkehrbar sein (in der Regel zur Erkennung von Unschärfen). Algorithmen von Codes, die umkehrbar sind, bestehen deshalb aus einem Codierer und einem Decodierer. Im Gegensatz dazu bestehen Algorithmen von nicht umkehrbaren Codes nur aus einem Codierer.

[169] Vgl. hierzu: Witt, K.-U.: Algebraische Grundlagen der Informatik; 2. Auflage; Vieweg Verlag; Wiesbaden; 2005; S. 209f
[170] Vgl. hierzu: Abschnitt 3.2.1.2 über Patterncodes
[171] Vgl. hierzu: Abschnitt 3.2.1.5 über Prüfzifferncodes

3.2.1.2 Patterncodes

Unter einem Patterncode soll die Umwandlung eines Wertes in ein allgemeines Muster verstanden werden. Patterncodes dienen der Vereinfachung von Zeichenketten, um Unregelmäßigkeiten in den Zeichenketten zu erkennen. Ein einfacher Algorithmus zur Erzeugung eines Patterncodes besteht darin, alle Zahlen durch eine ‚9' zu ersetzen, alle Buchstaben durch den Großbuchstaben ‚A' und die übrigen Zeichen zu belassen. Der Patterncode einer Telefonnummer „(040) 12345-112" würde dann lauten: „(999) 99999-999". Hat ein Attribut ein bestimmtes Darstellungsmuster, so könnte nun über eine Häufigkeitsverteilung überprüft werden, ob alle Werte dieses Attributes sich an das Eingabemuster halten.

Eine Möglichkeit, Eingabefehler oder Rechtschreibfehler zu erkennen, besteht in der Umformung von Zeichenketten. Hierzu werden Vertauschungen von Buchstaben oder auf der Tastatur nebeneinander liegende Zeichen berücksichtigt. J. Pollock und A. Zamora beschreiben zwei einfache Algorithmen („Skeleton Key" und „Omission Key" Algorithmus) zur Erzeugung eines Codes, um Eingabefehler zwischen verschiedenen Zeichenketten zu erkennen. Beim „Skeleton Key" Algorithmus bleibt das erste Zeichen erhalten, es folgen jeweils nur einmal die Konsonanten und danach die Vokale in der Reihenfolge ihres Auftretens.

Beispiel:

BUNTSTIFT = BNTSFUI
BUNTSTFIT = BNTSFUI

Das „Omission Key" Verfahren basiert dagegen auf Ergebnissen von Untersuchungen, nach denen bestimmte Konsonanten bei der Eingabe von Wörtern in folgender Reihenfolge eher weggelassen werden als andere:

RSTNLCHDPGMFBYWVZXQKJ

Das heißt, der Buchstabe ‚R' wird häufiger bei der Eingabe weggelassen als der Buchstabe ‚J'. Beim „Omission Key" Verfahren werden nun alle vorhandenen Konsonanten in dieser

Ordnung in umgekehrter Reihenfolge angeordnet. Die Vokale folgen wie beim „Skeleton Key“ Verfahren nach den Konsonanten in der Reihenfolge ihres Auftretens[172].

Beispiel:

KARAMELL = KMLRAE

KAMARELL = KMLRAE

3.2.1.3 Phonetische Codes

Algorithmen zur Erzeugung phonetischer Codes sollen im Idealfall für alle gleich ausgesprochenen Wörter einen gleichen Code erzeugen. Nachnamen wie Meyer, Maier oder Meier werden in der deutschen Sprache gleich ausgesprochen und sollten den gleichen phonetischen Code erhalten, um diesen „Gleichklang“ abzubilden. Dadurch ist es möglich, nach Wörtern zu suchen, bei denen die Aussprache und nicht die Schreibweise bekannt ist.[173] Wie die Aussprache von Wörtern sind auch phonetische Codes sprachabhängig und normalerweise nur für eine bestimmte Sprache geeignet.[174]

Der bekannteste phonetische Code ist der Soundex-Code, der 1900 zur Volkszählung in den USA von M. Odell und R. Russel entwickelt wurde. Beim Soundex-Algorithmus werden ähnlich klingende Buchstaben durch eine gleiche gemeinsame Zahl kodiert (z.B. ‚m’ und ‚n’ durch die Zahl 5). Mehrfach aufeinanderfolgende Konsonanten werden zu einem Konsonanten zusammengefasst, der erste Buchstabe bleibt immer erhalten und Vokale werden beim Code ignoriert. Schließlich wird der so ermittelte Code auf 4 Zeichen begrenzt. Ist er kürzer als 4 Zeichen, so wird er mit Nullen aufgefüllt.

[172] Pollock, J.J.; Zamora, A.: Automatic spelling correction in scientific and scholarly text; erschienen in: Communications of the ACM, Vol. 27, No. 4 (April 1984); New York; 1984; S. 358-368
[173] Wilz, M.: Aspekte der Kodierung phonetischer Ähnlichkeiten in deutschen Eigennamen; Magisterarbeit; Universität Köln, Philosophische Fakultät, Institut für Linguistik, Abteilung Phonetik; Köln; 2005; S. 4
[174] Vgl. hierzu: Pfeifer, U. u.a.: Searching Proper Names in Databases, University of Dortmund, Dortmund, 1994, S. 1

Tab. 6: Umsetzungsregeln des Soundex-Algorithmus

Buchstaben	Ziffern	Erzeugung der Laute
b, f, p, v	1	Lippen, öffnend
c, g, j, k, q, x, s, z	2	Rachen, Zunge
d, t	3	Zunge, Zähne
l	4	Zunge, Gaumen
m, n	5	Lippen, geschlossen
r	6	Zunge, rollend

Quelle: Scheibl, H.-J.: Grundlagen Phonetischer Datenbankabgleich; Fachhochschule für Technik und Wirtschaft Berlin; http://www2.f1.fhtw-berlin.de/scheibl/phonet/index.htm? ./grundl/grundl02.htm; abgerufen am 13.08.2007

Die Nachnamen Meyer, Maier und Meier ergeben als Soundex-Code „M600“, haben also aufgrund der gleichen Aussprache die gleiche Kodierung. Allerdings hat auch der Nachname Mohr den Soundex-Code „M600“. Dies ist möglich, weil der Soundex-Code für die englische Sprache entwickelt wurde und weil der Algorithmus zu trivial ist. Der Algorithmus betrachtet jeden Buchstaben einzeln und keine Folgen von Buchstaben bei der Aussprache. Aus dieser Kritik heraus wurden deshalb Algorithmen entwickelt, die auch Buchstabengruppen und die Lage dieser innerhalb des Wortes (Anfang, Mitte, Ende) berücksichtigen.

Hierzu gehört der NYSIIS-Algorithmus (New York State Identification and Intelligence System) von 1970, der auch Buchstabengruppen berücksichtigt[175]. Gegenüber dem Soundex-Code beträgt der Präzisionsgewinn 2,7%.[176]

Eine Kombination aus NYSIIS- und Soundex-Code ist der ONCA-Code (Oxford Name Compression Algorithm), der im Rahmen der „Oxford Record Linkage Studie“ (ORLS) zum Blocking bei der Erkennung von Duplikaten eingesetzt wurde.[177] ONCA ist ein zweistufiges

[175] Newcombe, H.: Handbook of Record Linkage, Oxford University Press, New York, 1988, S. 182f
[176] Elmagarmid, A.K.; Ipeirotis, P.G.; Verykios, V.S.: Duplicate Record Detection: A Survey; erschienen in: IEEE Transactions on Knowledge and Data Engineering, Vol. 19, No. 1; IEEE Computer Society; Washington, DC; 2007; S. 5
[177] Gill, L.: Methods for Automatic Record Matching and Linkage and their Use in National Statistics, National Statistics Methodological, Series No. 25; National Statistics; London; 2001; S. 110

Verfahren: Zunächst wird eine britische Variante des NYSIIS-Algorithmus verwendet, um im zweiten Schritt den erzeugten Code über den Soundex-Algorithmus zu kodieren.[178]

Der 1970 entwickelte „IBM Alpha Inquiry System Personal Name Encoding“ Code ist 14 Zeichen lang und besteht nur aus Zahlen.[179]

In einer Erweiterung des Soundex-Codes von R. Daitch und G. Mokotoff werden besonders die phonetischen Eigenschaften von Namen osteuropäisch-jüdischer Einwanderer implementiert.[180] Der Daitch-Mokotoff-Algorithmus unterstützt Buchstabengruppen und die Möglichkeit, mehrere Codes mit einer Länge von 6 Zeichen zu erzeugen.[181]

Der Phonix-Algorithmus transformiert etwa 160 Buchstabengruppen[182] und erweitert die Codelänge auf 8 Zeichen. Er wurde von T. Gadd entwickelt und ist wie der Soundex-Code numerisch. Ist der erste Buchstabe ein Vokal, so wird dieser durch das $-Zeichen ersetzt.[183]

L. Philips entwickelte 1990 erstmals einen phonetischen Code mit Namen Metaphone, der wesentlich mehr Regeln enthält als die bisher beschriebenen Verfahren und den Code auf 16 Konsonanten reduziert.[184] Dabei werden die Zeichen nicht einzeln wie beim Soundexverfahren, sondern immer in ihrem phonetischen Kontext betrachtet.[185] Im Jahr 2000 verbesserte L. Philips den Algorithmus und bezeichnete ihn aufgrund der Erzeugung von zwei Codes (Primary und Alternate) als Double Metaphone. Bei der Überprüfung, ob zwei Nachnamen phonetisch übereinstimmen, ist damit eine bessere Abstufung möglich: Nur der Primary Code, nur der Alternate Code oder beide stimmen überein. Zudem werden bei diesem

[178] Elmagarmid, A.K.; Ipeirotis, P.G.; Verykios, V.S.: Duplicate Record Detection: A Survey; erschienen in: IEEE Transactions on Knowledge and Data Engineering, Vol. 19, No. 1; IEEE Computer Society; Washington, DC; 2007; S. 5
[179] Armstrong, M.: An Overview of the Issues Related to the Use of Personal Identifiers, Catalogue no. 85-602-XIE; Statistics Canada; Ottawa; 2000; S. 16
[180] Heller, M.: Approximative Indexierungstechniken für historische deutsche Textvarianten; erschienen in: Historical Social Research / Historische Sozialforschung, Vol. 31 (2006), No. 3; Zentrum für Historische Sozialforschung e.V.; Köln; 2006; S. 296
[181] Wilz, M.: Aspekte der Kodierung phonetischer Ähnlichkeiten in deutschen Eigennamen; Magisterarbeit; Universität Köln, Philosophische Fakultät, Institut für Linguistik, Abteilung Phonetik; Köln; 2005; S. 16
[182] Zobel, J.; Dart, P.: Phonetic String Matching: Lessons from Information Retrieval, University of Melbourne, Melbourne, o.J., S. 2
[183] Wilz, M.: Aspekte der Kodierung phonetischer Ähnlichkeiten in deutschen Eigennamen; Magisterarbeit; Universität Köln, Philosophische Fakultät, Institut für Linguistik, Abteilung Phonetik; Köln; 2005; S. 14
[184] Philips, L.: Hanging on the Metaphone; erschienen in: Computer Language Magazine, Vol. 7, No. 12, 1990, S. 38f
[185] Heller, M.: Approximative Indexierungstechniken für historische deutsche Textvarianten; erschienen in: Historical Social Research / Historische Sozialforschung, Vol. 31 (2006), No. 3; Zentrum für Historische Sozialforschung e.V.; Köln; 2006; S. 297

Algorithmus auch Nachnamen anderer Landessprachen in ihrer amerikanischen Aussprache berücksichtigt.[186] Der Code besteht dabei aus Buchstaben und ist variabel lang.[187]

Ein am Soundex-Code angelehnter Algorithmus für den deutschsprachigen Bereich ist der Kölner Phonetik Code von H.-J. Postel aus dem Jahr 1969.[188] Er berücksichtigt wie der Soundex-Code nur einzelne Buchstaben und keine Buchstabengruppen.

Ein weiterer einfacher phonetischer Code für deutsche Namen stammt von H.-P. von Reth und H.-J. Schek aus dem Jahr 1977,[189] der einfache Buchstabengruppen unterstützt.

Der PHONEM-Algorithmus von G. Wilde und C. Meyer aus dem Jahr 1988 arbeitet in zwei Schritten. Zunächst werden Gruppen von jeweils zwei Buchstaben und im zweiten Schritt einzelne Buchstaben transformiert.[190]

Die aktuellste Entwicklung eines phonetischen deutschsprachigen Codes stammt von J. Michael aus dem Jahr 1988[191] und eine erweiterte Version von 1999[192]. Enthielt die erste Version des phonet-Algorithmus noch etwa 30 Regeln, so verwendet der aktuelle mehr als 920 Regeln zur Code-Erzeugung und ist damit wesentlich komplexer als die drei anderen erwähnten deutschsprachigen phonetischen Codes.

Da die Aussprache von Wörtern sprachabhängig ist, sind phonetische Codes auch sprachen-abhängig[193]. Nach J. Zobel und P. Dart sind phonetische Algorithmen beim Vergleich von zwei Zeichenketten bei weitem nicht so effektiv wie „Approximate String Matching" Algorithmen.[194] Das generelle Problem bei phonetischen Algorithmen ist, dass letztendlich

[186] Philips, L.: The Double Metaphone Search Algorithm, C/C++ Users Journal, Vol. 18 No. 6, Boulder, 2000
[187] Wilz, M.: Aspekte der Kodierung phonetischer Ähnlichkeiten in deutschen Eigennamen; Magisterarbeit; Universität Köln, Philosophische Fakultät, Institut für Linguistik, Abteilung Phonetik; Köln; 2005; S. 14
[188] Schürle, J.: Record Linkage, Peter Lang Verlag, Tübingen, 2004, S. 44
[189] Heller, M.: Approximative Indexierungstechniken für historische deutsche Textvarianten; erschienen in: Historical Social Research / Historische Sozialforschung, Vol. 31 (2006), No. 3; Zentrum für Historische Sozialforschung e.V.; Köln; 2006; S. 298
[190] Wilz, M.: Aspekte der Kodierung phonetischer Ähnlichkeiten in deutschen Eigennamen; Magisterarbeit; Universität Köln, Philosophische Fakultät, Institut für Linguistik, Abteilung Phonetik; Köln; 2005; S. 18
[191] Michael, J.: Nicht wörtlich genommen - Schreibweisentolerante Suchroutinen in dBase implementiert; erschienen in: c't Magazin für Computer und Technik 10/1988. S. 126ff
[192] Michael, J.: Doppelgänger gesucht - Ein Programm für kontextsensitive phonetische Stringumwandlung; erschienen in: c't Magazin für Computer und Technik 25/1999. S. 252ff
[193] Pfeifer, U.; Poersch, T.; Fuhr, N.: Retrieval Effectiveness of Proper Name Search Methods; erschienen in: Information Processing and Management, Vol. 32, No. 6, S. 668
[194] Zobel, J.; Dart, P.: Finding Approximate Matches in Large Lexicons; erschienen in: Software - Practice and Experience, Vol. 25, Issue 3 (March 1995); John Wiley & Sons; New York, USA; S. 344

jede Eigenschaft bei der Aussprache durch komplexe Regeln abgebildet werden muss. Hinzu kommt, dass die Aussprache sich ändern kann und die Aussprache in verschiedenen Sprachen unterschiedlich ist. So wird der Vorname „Michael“ im Deutschen, Englischen und Russischen jeweils völlig unterschiedlich ausgesprochen. Die hier betrachteten phonetischen Codes sind daher auch nur für Eigennamen und nur für die Sprache geeignet, für die sie entwickelt wurden.[195] Ziel eines phonetischen Algorithmus sollte die Abbildung einer Zeichenkette in einen Code sein, der für alle gleich ausgesprochenen Wörter identisch ist. Mit den bisher entwickelten Algorithmen ist dieses Ziel bisher nicht erreicht, da der Code sprachabhängig ist. Selbst bei Berücksichtigung der gleichen Sprache existieren Ambiguitäten der Aussprache durch z.B. Dialekte, die Vermischung der Kulturen und unterschiedliche Aussprachen zu unterschiedlichen Zeiten.[196] Zudem könnten nach M. Wilz Verbesserungen erreicht werden, indem wesentliche Merkmale eines Wortes am Anfang des Codes erscheinen und unwesentliche am Ende.[197]

Für Anwendungen jedoch, bei denen der Recall-Wert eine größere Bedeutung als der Precision-Wert hat,[198] sind phonetische Codes durchaus geeignet. Für das Analysieren von Daten auf Grundlage von Häufigkeitsverteilungen und für das Partitionieren[199] von Daten ist der Einsatz phonetischer Codes daher sinnvoll.

3.2.1.4 Prüfzifferncodes

Prüfzifferncodes werden z.B. bei der ISBN (International Standard Book Number) und bei der EAN (European Article Number) verwendet.[200] Dabei wird der eigentlichen Zeichenfolge in der Regel ein zusätzliches Zeichen hinzugefügt, um hierüber die Korrektheit der Eingabe zu überprüfen. Algorithmen zur Erzeugung solcher Prüfzifferncodes dienen der Erkennung von Fehlern bei der Eingabe. Dabei wird normalerweise nicht automatisch korrigiert, sondern es

[195] Vgl. hierzu: Wilz, M.: Aspekte der Kodierung phonetischer Ähnlichkeiten in deutschen Eigennamen; Magisterarbeit; Universität Köln, Philosophische Fakultät, Institut für Linguistik, Abteilung Phonetik; Köln; 2005; S. 2
[196] Vgl. hierzu den Caverphone-Algorithmus aus dem Caversham Projekt der Universität Otaga, Neuseeland, der auf einen bestimmten Datenbestand mit Namen von Personen aus den Jahren 1893 bis 1938 angepasst wurde: Hood, D.: Phonetic Matching Algorithm, Caversham Technical Paper, CTP060902; University of Otago; Otaga, Neuseeland, 2002
[197] Wilz, M.: Aspekte der Kodierung phonetischer Ähnlichkeiten in deutschen Eigennamen; Magisterarbeit; Universität Köln, Philosophische Fakultät, Institut für Linguistik, Abteilung Phonetik; Köln; 2005; S. 7f
[198] zu Recall und Precision siehe Abschnitt 3.5.3.5 Metriken zur Überprüfung
[199] Vgl. hierzu Abschnitt: 3.4.3.2 Reduktion des Suchraums
[200] Witt, K.U.: Algebraische Grundlagen der Informatik; 2. Auflage; Vieweg Verlag; Wiesbaden; 2005; S. 267

muss eine erneute Eingabe erfolgen. Prüfziffern werden nach einem bestimmten Algorithmus aus den vorgegebenen Zeichen errechnet und können durch erneute Berechnung bei der Eingabe überprüft werden. Es werden hierfür also keine zusätzlichen Daten aus z.B. Nachschlagetabellen benötigt.

Prüfzifferncodes sind hauptsächlich zur Fehlererkennung und nicht zur Fehlerkorrektur gedacht.[201] Ergibt die Berechnung zur Überprüfung der Prüfziffer, dass diese falsch ist, dann sind die Eingabedaten nicht korrekt. Es kann bei einer korrekten Prüfziffer aber trotzdem vorkommen, dass ein oder mehrere Fehler vorhanden sind.[202] Ein Prüfziffernverfahren wird deshalb danach bewertet, wie hoch seine Fehlererkennungsrate ist. Die Qualität von Algorithmen zur Erzeugung von Prüfzifferncodes wird normalerweise auf der Grundlage einer empirischen Untersuchung von J. Verhoeff[203] bewertet, der verschiedene Fehlertypen identifiziert hat, die durch die Algorithmen erkannt werden sollten. Diese für sechsstellige Zahlen durchgeführte Untersuchung[204] zeigt, dass vor allem Einzelfehler – also die Eingabe eines einzelnen falschen Zeichens – am häufigsten auftritt (60-95%), gefolgt von Vertauschungen benachbarter oder übernächster Zeichen (11%).[205] Um die Qualität eines Algorithmus zu beurteilen, ist es sinnvoll zu überprüfen, wie hoch die Erkennungsrate bezogen auf diese Fehlertypen ist.

[201] Vgl. hierzu: Schulz, R.-H.: Codierungstheorie; 2. Auflage; Vieweg Verlag; Wiesebaden; 2003; S. 56
[202] Wagner, N. R.; Putter, P. S.: Error Detecting Decimal Digits; erschienen in: Communications of the ACM, Vol. 32, No. 1 (January 1989); ACM Press; New York, USA; 1989; S. 106
[203] Verhoeff, J.: Error Detecting Decimal Codes; 2. Auflage; Schriftenreihe Mathematical Centre Tracts No. 29; Mathematical Centre; Amsterdam; 1975; S. 10ff
[204] In „Damm, H. M.: Total anti-symmetrische Quasigruppen; Phillips-Universiät, Fachbereich Mathematik und Informatik; Marburg/Lahn; 2004, S. 11“ wird von einer aktuellen Untersuchung von Damm berichtet, die ähnliche Ergebnisse wie die von Verhoeff lieferte.
[205] Die Untersuchung wurde für sechsstellige Zahlen Ende der sechziger Jahre durchgeführt. Die Werte sind somit eher als Richtwerte zu verstehen und sollen eher zeigen, dass vor allem Einzelfehler und Nachbartranspositionen in einem Prüfziffernverfahren berücksichtigt werden sollten. Es leuchtet ein, dass mit zunehmender Stellenzahl auch mehr Fehler auftreten (vgl. hierzu: Michael, Jörg: Mit Sicherheit – Prüfziffernverfahren auf Modulo-Basis; erschienen in: c’t magazin für computertechnik; Ausgabe 7/97, S. 264ff)

Tab. 7: Fehlertypen

Fehlertyp	Beispiel	Anteil
Einzelfehler (eine falsche Ziffer)	*1* eingegeben als *2*	79% (60-95%)
Nachbartransposition (Transposition mit benachbarter Ziffer)	*12* eingegeben als *21*	10,2% (10-20%)
Sprungtransposition (Transposition mit anderer Ziffer)	*123* eingegeben als *321*	0,8% (0,5-2,5%)
Sonstige Fehler (Zwillingstransposition, phonetische Fehler, Sprung-Zwillingsfehler u.a.)	*11* eingegeben als *22* *15* eingegeben als *50* *191* eingegeben als *292*	10%

Quelle: eigene Darstellung in Anlehung an: Wagner, N. R.; Putter, P. S.: Error Detecting Decimal Digits; erschienen in: Communications of the ACM; January 1989, Vol. 32, No. 1; ACM Press; New York, USA; 1989; S. 108

Michael, Jörg: Mit Sicherheit – Prüfziffernverfahren auf Modulo-Basis; erschienen in: c't magazin für computertechnik; Ausgabe 7/97, S. 264ff

Ein einfacher Prüfziffernalgorithmus bildet z.B. die Quersumme und fügt eine Zahl hinzu, so dass die neue Ziffernfolge eine Zehnerzahl ergibt. Besteht eine Zeichenkette aus den Ziffern 123456, lautet die Prüfziffer nach diesem Verfahren die Zahl ‚9', denn die Quersumme von 1+2+3+4+5+6 ergibt 21, so dass die Ziffer ‚9' zum Erreichen der nächsten Zehnerzahl (‚30') der Ziffernfolge hinzugefügt werden muss (1+2+3+4+5+6+9). Dieser einfache Algorithmus erkennt zwar Einzelfehler, aber keine Vertauschungen von einzelnen Zeichen. Um zusätzlich Falscheingaben benachbarter Zeichen zu erkennen, wird jeder Stelle innerhalb der Zeichenkette ein Gewicht zugeordnet.

Die bekanntesten Prüfziffernalgorithmen haben alle ein ähnliches Schema. Generell wird aus allen Zeichen die Quersumme gebildet, nachdem die einzelnen Ziffern mit einem Gewichtungsfaktor multipliziert wurden. Danach wird dieser Wert mittels modularer Arithmetik durch eine Konstante dividiert und der Modulo als Ergebnis an die originale Zeichenfolge angehängt. Sind alle Gewichte teilerfremd zum Modulo, so lassen sich alle Einzelfehler erkennen. Sind außerdem die Differenzen aufeinanderfolgender Gewichte

teilerfremd zum Modulo, so werden auch einzelne Nachbartranspositionen erkannt.[206] Die bekanntesten Verfahren verwenden als Modulo die Zahlen 7, 9, 10 und 11.

Das folgende Beispiel zeigt die Berechnung der Prüfziffer für eine ISBN-10, bei der als Gewichte die Zahlen von 10 bis 2 verwendet werden und als Modulo die Zahl 11:

Tab. 8: Beispiel zur Berechnung des ISBN-Prüfzifferncodes

ISBN-10	3	8	2	7	3	1	9	3	8
Gewicht	10	9	8	7	6	5	4	3	2
Produkt	30	72	16	49	18	5	36	9	16
Summe	251								
Modulo 11	9								
Prüfziffer	11 – 9 = 2								

Quelle: eigene Darstellung

Die vollständige ISBN-10 lautet im Beispiel 3827319382. Bei einem Divisionsrest von 1 würde sich allerdings die zweistellige Prüfziffer 10 ergeben. Dieses Problem wird bei der ISBN dadurch gelöst, dass die römische Zahl ‚X' als Prüfziffer verwendet wird. Da es sich bei der Zahl 11 um eine Primzahl handelt und alle Gewichte kleiner als 11 sind, lassen sich mit diesem Prüfzifferverfahren alle Einzelfehler erkennen. Außerdem werden alle Transpositionsfehler von zwei beliebigen Ziffern erkannt.[207] Nicht erkannt werden dagegen ein Teil der Zwillingstranspositionen und phonetischen Fehler.[208]

Daneben existieren Derivate, wie z.B. der Luhn- bzw. „IBM check"-Algorithmus, der zur effizienteren Fehlererkennung die Quersumme aus den Einzelprodukten von Ziffer und Gewichten bildet. Dieser bei Kreditkarten eingesetzte Algorithmus erkennt alle Einzelfehler und alle Transpositionen benachbarter Zeichen (außer 09 und 90).[209] Der Luhn-Algorithmus

[206] Vgl. hierzu: Michael, Jörg: Mit Sicherheit – Prüfziffernverfahren auf Modulo-Basis; erschienen in: c't magazin für computertechnik; Ausgabe 7/97, S. 264ff
[207] Wagner, N. R.; Putter, P. S.: Error Detecting Decimal Digits; erschienen in: Communications of the ACM; Vol. 32, No. 1 (January 1989); ACM Press; New York, USA; 1989; S. 106
[208] Vgl. hierzu: Michael, Jörg: Mit Sicherheit – Prüfziffernverfahren auf Modulo-Basis; erschienen in: c't magazin für computertechnik; Ausgabe 7/97, S. 264ff
[209] Wagner, N. R.; Putter, P. S.: Error Detecting Decimal Digits; erschienen in: Communications of the ACM, Vol. 32, No. 1 (January 1989); ACM Press; New York, USA; 1989; S. 107

erkennt damit 88 von 90 möglichen Nachbartranspositionen, also 97,8%.[210] Die höchste Fehlererkennungsrate bietet die Verwendung von Modulo 11 mit Gewichten, die der Reihenfolge der Potenz von 2 entsprechen.[211]

Daneben wurden komplexere Verfahren wie der Verhoeff-Algorithmus entwickelt, der zur Berechnung der Prüfziffer sog. Diëdergruppen[212] verwendet. Gegenüber dem ISBN- hat der Verhoeff-Algorithmus den Vorteil, dass die Prüfziffer zwischen 0 und 9 liegt und die Eingabedaten beliebig lang sein können.[213] Der Verhoeff-Algorithmus erkennt neben allen Einzelfehlern auch alle Nachbartranspositionen und 94,2% von Sprungtranspositionen.[214]

Modulo 7 und 9 werden zwar oft zur Berechnung von Prüfziffern verwendet (United Parcel Service Pakete, Federal Express Mail u.a.), sind aber nicht so effizient wie Modulo 11, da nicht einmal alle Einzelfehler erkannt werden.[215]

[210] Gallian, J.A.: Error Detection Methods; erschienen in: ACM Computing Surveys; September 1996, Vol. 28, No. 3; ACM Press; New York, USA; 1996; S. 506f
[211] Vgl. hierzu: Michael, Jörg: Mit Sicherheit – Prüfziffernverfahren auf Modulo-Basis; erschienen in: c't magazin für computertechnik; Ausgabe 7/97, S. 264ff
[212] Vgl. hierzu: Witt, K.-U.: Algebraische Grundlagen der Informatik; 2. Auflage; Vieweg Verlag; Wiesbaden; 2005; S. 278ff
[213] Vgl. hierzu: Kirtland, J.: Identification Numbers and Check Digit Schemes; The Mathematical Association of America; 2001; S. 153
[214] Wagner, N. R.; Putter, P. S.: Error Detecting Decimal Digits; erschienen in: Communications of the ACM, Vol. 32, No. 1 (January 1989); ACM Press; New York, USA; 1989; S. 108
[215] Gallian, J.A.: Error Detection Methods; erschienen in: ACM Computing Surveys; September 1996, Vol. 28, No. 3; ACM Press; New York, USA; 1996; S. 510

Tab. 9: Fehlertypenerkennung von Prüfziffernverfahren

Algorithmus	**Einzel-fehler**	**Transpositionen**				**Phonet. Fehler**	**Sonst. Fehler geschätzt**
		Nachbar	**Sprung**	**Zwilling**	**Sprung-Zwilling**		
Modulo 10 Gewichte 1,1,1…	100,0	0,0	0,0	88,9	88,9	100,0	90,0
Modulo 11 ISBN	100,0	100,0	100,0	80,0	100,0	88,9	90,9
Modulo 11 Gewichte 2^i	100,0	100,0	100,0	100,0	100,0	100,0	90,9
Verhoeff Diedergruppe D_{10}	100,0	100,0	94,2	95,6	94,2	100,0	90,0

Quelle: eigene Darstellung in Anlehnung an: Gallian, J.A.: Error Detection Methods; erschienen in: ACM Computing Surveys; September 1996, Vol. 28, No. 3; ACM Press; New York, USA; 1996; S. 515
Michael, Jörg: Mit Sicherheit – Prüfziffernverfahren auf Modulo-Basis; erschienen in: c't magazin für computertechnik; Ausgabe 7/97, S. 264ff

Daneben existieren Algorithmen, die mit mehr als einer Prüfziffer Fehler automatisch korrigieren können. J.A. Gallian beschreibt ein Verfahren zur Berechnung von zwei Prüfziffern auf Modulo 11-Basis, das alle Einzelfehler automatisch korrigieren kann. Über die erste Prüfziffer wird dabei die Differenz des Einzelfehlers ermittelt und über die zweite die Stelle der Ziffer, die inkorrekt ist.[216]

3.2.1.5 Hashingcodes

Über Hashingcodes können Daten partitioniert und als kleinere Menge von Daten abgebildet werden.[217] Hashingcodes werden ursprünglich vor allem beim Suchen in Datenbanken verwendet, indem der Code als Schlüssel verwendet wird. Geht man bei einem Attribut davon aus, dass es eine große Menge von potenziellen Schlüsseln gibt, ist die tatsächlich verwendete Menge an Schlüsseln normalerweise relativ klein. Über eine Hashingfunktion[218] wird nun der Wert eines Attributes in einen Schlüsselwert transformiert und in einer Hashtabelle

[216] Vgl. hierzu: Gallian, J.A.: Error Detection Methods; erschienen in: ACM Computing Surveys; September 1996, Vol. 28, No. 3; ACM Press; New York, USA; 1996; S. 511f
[217] Vgl. hierzu: Dasu, T.; Johnson, T.: Exploratory Data Mining and Data Cleaning; John Wiley & Sons; Hoboken, New Jersey; 2003; S. 171
[218] Zum Hashing siehe: Sedgewick, R.: Algorithmen, 2. Auflage; Addison-Wesley; München; 2002; S. 273ff

gespeichert, um ihn über die Hashingfunktion direkt zu adressieren. Hierzu ein Beispiel: Eine Tabelle „Nachname“ soll über Hashcodes gespeichert und durchsucht werden. Die Hashtabelle soll für 11 Einträge vorgesehen sein, so dass über eine geeignete Hashfunktion möglichst eine direkte Adressierung zu jedem gesuchten Wert erfolgt. Als Hashfunktion wird zunächst die Länge der Zeichenkette des Nachnamens verwendet. Dieser Wert entspricht dann dem Index in der Hashtabelle. Für die Nachnamen in Tabelle 10 wurden folgende Hashcodes errechnet:

Tab. 10: Beispiel zur Berechnung von Hashcodes

Index	**Länge Zeichenkette**	**Quersumme Buchstaben mit wiederholenden Gewichten (5, 7, 9, 11) mod 11**
0		
1		Hansen
2		Meier
3		Kolle
4	Kohl	Kohl
5	Cortz, Kolle, Maier, Meier, Meyer	
6	Cordts, Hansen	Cortz
7		Cordts
8		Meyer
9		
10		Maier

Quelle: eigene Darstellung

Bei diesem Beispiel kommt es zu 5 Kollisionen, bei denen der gleiche Index bzw. Hashcode verwendet wird. Ziel bei der Verwendung von Hashtabellen ist es, eine Hashfunktion zu finden, die möglichst wenig Kollisionen verursacht, so dass ein direktes Finden eines Datensatzes über die Funktion möglich ist. Im zweiten Beispiel wird eine etwas komplexere Hashfunktion mit Gewichten verwendet, so dass keine Kollisionen mehr auftreten (siehe Tabelle 10).

Da es bei Datenqualitätsproblemen darauf ankommt, Unschärfen in den Daten zu finden, sind Kollisionen bei Hashingverfahren im Datenqualitätsbereich durchaus gewollt. Abhängig von der Kollisionsfunktion ist es möglich, Werte zu gruppieren, die ähnliche Eigenschaften besitzen.

3.2.1.6 Stemming

Um Ähnlichkeiten zwischen verschiedenen Texten zu erkennen, sollten Wörter auf ihre ursprüngliche Wortform zurückgeführt werden, um sie als Wort mit gleichem Wortstamm erkennen zu können (z.B. gehen als Grundform für ging, gegangen, gehe). Das automatische Zusammenführen und Gruppieren zu solchen Wortformen wird als Stemming[219] oder Lemmatisierung bezeichnet. Das Konzept des Stemming stammt aus den 60er Jahren und hatte das ursprüngliche Ziel, die Anzahl unterschiedlicher Wörter in Texten zu reduzieren, um die Performanz zu verbessern. Ein weiterer Vorteil des Stemming besteht in der Verbesserung des Recall.[220]

Beim Stemming unterscheidet man zwischen Stamm- und Grundformenreduktion. Die Grundform eines Wortes ist ihre grammatikalische Grundform (z.B. aufgenommen in aufnehmen, Aufnahmen in Aufnahme). Auf die Stammform können dagegen sowohl Substantive als auch Verben zurückgeführt werden. Die Stammform ist normalerweise kein Wort, das im allgemeinen Sprachgebrauch verwendet wird (z.B. aufnehmen in aufnehm).[221] Verfahren zur Reduktion auf die Stammform werden manchmal als Stemming und Verfahren zur Reduktion auf die Grundform als Lemmatisierung bezeichnet. Stemming und Lemmatisierung findet man auch gelegentlich unter dem Oberbegriff Conflation Verfahren. Da sich die Unterscheidung der Begriffe Stemming, Lemmatisierung und Conflation in der Form nicht durchgesetzt hat und in der Literatur meist der Begriff Stemming für alle Verfahren zur Reduktion eines Wortes verwendet wird, soll zur Vereinfachung auch hier dieser Definition gefolgt werden.

W. B. Frakes unterscheidet vier Arten von Stemmingverfahren:

- Nachschlagetabellen
- Affix Removal
- N-Grams
- Successor Variety

[219] engl.: to stem = abstammen von
[220] Kowalski, G.J.; Maybury, M.T.: Information Storage and Retrieval Systemes - Theory and Implementation, Second Edition; Kluwer Academic; Norwell, Massachusetts; 2000; S. 73f
[221] Ferber, R.: Information Retrieval; dpunkt Verlag; Heidelberg; 2003; S. 40f

Eine einfache Form des Stemmings besteht in der Verwendung einer Nachschlagetabelle, in der für jedes Wort dessen Grund- oder Stammform aufgeführt wird.

Die am häufigsten eingesetzte Methode des Stemmings besteht im Entfernen von Präfixen und Suffixen (Affixen) des Wortes.[222] Der bekannteste ursprünglich für die englische Sprache entwickelte Stemming-Algorithmus, der Affixe entfernt, stammt von M.F Porter. In einem fünfstufigen Algorithmus werden hier um die 60 Suffixe von Wörtern entfernt.[223] Inzwischen wurde dieser Algorithmus um 14 weitere Sprachen ergänzt.[224] Der erste Stemming-Algorithmus wurde 1968 von J. Lovins entwickelt. Er entfernt mehr als 260 unterschiedliche Suffixe aus englischen Wörtern.[225] Das auch für englische Wörter entwickelte Stemming-Verfahren von C. Paice und G. Husk verwendet Regeln, die das Löschen und Ersetzen von Wortendungen beschreiben.[226] Da diese Art der Stemming-Algorithmen ohne Berücksichtigung der Semantik Suffixe entfernen, kann die zurückgeführte Grundform dem Wort eine andere Bedeutung geben (der Lovins-Algorithmus führt z.B. das englische Wort „army" auf die Grundform „arm" zurück). In der Bedeutung unterschiedliche Wörter können damit fälschlicherweise eine gemeinsame Grundform erhalten. R. Krovetz entwickelte deshalb einen Algorithmus, der sowohl Nachschlagetabellen verwendet, aber auch Suffixe entfernt.[227] Ein ähnlicher Ansatz findet sich auch bei dem UEA (University of East-Anglia)-Lite Stemmer, der Wörter auf ihr orthographisch korrektes Grundwort zurückführt und Wörter wie Eigennamen erkennt, die nicht verändert werden.[228] Ein weiterer Stemming-Algorithmus für deutsche Wörter stammt von J. Caumann.[229]

Ein Verfahren von G.W. Adamson und J. Boreham (1974) verwendet N-Grams der Größe 2, um einzelne Wörter auf ein gemeinsames Wort zurückführen, indem über das Dice-Ähnlichkeitsmaß zwischen allen Wörtern in der Zeichenkette die Ähnlichkeit zwischen diesen

[222] Manning, C. D., Schütze, H.: Foundations of Statistical Natural Language Processing; The MIT Press; Cambridge, Massachusetts; 2003; S. 132
[223] Porter, M. F.: An algorithm for suffix stripping; erschienen in: Sparck Jones, K.; Willett, P (Hrsg.).: Readings in Information Retrieval; Morgan Kaufmann Publishers; San Francisco; 1997; S. 313ff
[224] Siehe hierzu: http://snowball.tartarus.org
[225] Krovetz, R.: Viewing Morphology as an Inference Process; erschienen in: Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval; 1993; S. 191
[226] Paice, C. D.: Another stemmer; erschienen in: SIGIR Forum 24/3; S. 56–61
[227] Vgl. hierzu: Krovetz, R.: Viewing Morphology as an Inference Process; erschienen in: Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval; 1993; S. 191-203
[228] Vgl. hierzu: Jenkins, M.-C.; Smith, D.: Conservative stemming for search and indexing; erschienen in: Proceedings of the 28.th ACM SIGIR Conference; Salvador, Brasilien; 2005
[229] Caumanns, J.: A Fast and Simple Stemming Algorithm; Technical Report Nr. TR-B-99-16 des Fachbereichs Informatik der Freien Universität Berlin; 1999

ermittelt wird.[230] Bei N-Grams wird die Zeichenkette in gleich große Teilzeichenketten, hier der Größe 2, aufgeteilt.

M.A. Hafer und S.F. Weis beschreiben in „Word Segmentation by Letter Successor Variety" ein Verfahren, das die Anzahl von Nachfolgebuchstaben der einzelnen Wörter in einer Zeichenkette ermittelt.[231] Besteht eine Zeichenkette aus den Wörtern „Trompete, Triumpf, Tasse, Teller", so ergeben sich als Nachfolgevarietäten (Successor Variety) für das Wort „Trinken" im ersten Schritt drei Nachfolger (auf den Buchstraben ‚T' folgen drei unterschiedliche Zeichen: ‚r', ‚a' und ‚e'). Im zweiten Schritt ergibt sich eine Varietät von 2 (Buchstaben ‚o' und ‚i') und im letzen von 1 (Buchstabe ‚u' im Wort „Triumpf"). Normalerweise sinkt die Varietät zunächst, um dann an einer Wortgrenze wieder anzusteigen. Diese Eigenschaft wird ausgenutzt, um Wörter in Einzelwörter aufzuteilen. Dabei werden vier verschiedene Arten zur Segmentierung vorgeschlagen.[232]

Da durch das Stemming Informationen verloren gehen, erhöht sich durch das Stemming generell der Recall und die Precision sinkt.[233]

3.2.1.7 Stoppwörter

Im Bereich des Information Retrieval erkannte man bereits Ende der 50er Jahre, dass im Englischen häufig auftretende Wörter nicht für die Suche nach Dokumenten geeignet sind, da die Suche nach einem häufig auftretenden Wort meistens alle Einträge in der Datenbasis zurückliefert.[234] Bei einem Vergleich von zwei Zeichenketten, die aus längeren Texten bestehen, kann es deshalb sinnvoll sein, zunächst Wörter herauszufiltern, die eine sehr hohe

[230] Frakes, W.B.: Stemming Algorithms; erschienen in: Frakes, W.B.; Baeza-Yates, R. (Hrsg.): Information Retrieval - Data Structures & Algorithms; Prentice-Hall; Upper Saddle River, New Jersey; 1992; S. 136
[231] Vgl. hierzu: Hafer, M. A.; Weis, S. F.: Word Segmentation by Letter Successor Variety; erschienen in: Information Storage and Retrieval, Vol. 10, Issue 11-12; Pergamon Press; Oxford; 1974; S. 371-385
[232] Frakes, W.B.: Stemming Algorithms; erschienen in: Frakes, W.B.; Baeza-Yates, R. (Hrsg.): Information Retrieval - Data Structures & Algorithms; Prentice-Hall; Upper Saddle River, New Jersey; 1992; S. 134
[233] Frakes, W.B.: Stemming Algorithms; erschienen in: Frakes, W.B.; Baeza-Yates, R. (Hrsg.): Information Retrieval - Data Structures & Algorithms; Prentice-Hall; Upper Saddle River, New Jersey; 1992; S. 150
[234] Fox, C.: Lexical Analysis and Stoplists; erschienen in: Frakes, W.B.; Baeza-Yates, R. (Hrsg.): Information Retrieval - Data Structures & Algorithms; Prentice-Hall; Upper Saddle River, New Jersey; 1992; S. 113

Häufigkeit in allen Zeichenketten haben[235] und die damit für den Vergleich nicht relevant sind. Diese herauszufilternden Wörter bezeichnet man als Stoppwörter.[236]

Allgemeine Stoppwortlisten der deutschen Grammatik enthalten u.a. bestimmte und unbestimmte Artikel („der“, „die“, „einer“), Konjunktionen („und“, „damit“) usw. Für domänenspezifische Vergleiche kann eine Stoppwortliste aber durchaus auch Domänenbegriffe enthalten, die für einen Vergleich nicht relevant sind. So kann es bei einem Vergleich von Namen z.B. sinnvoll sein, Titel als Stoppwörter herauszulöschen.

Das Herausfiltern von Stoppwörtern kann allerdings auch bewirken, dass der Recall sich verringert. Phrasen, die nur aus Stoppwörtern bestehen (z.B. „Sein oder Nichtsein“) werden danach komplett herausgefiltert.[237] Handelt es sich bei Stoppwörtern um Homonyme zu selten vorkommenden Substantiven (z.B. „fest“ und „Fest“), so kann es vorkommen, dass auch selten vorkommende Wörter herausgefiltert werden.[238]

3.2.1.8 Codelisten

Werden unterschiedliche Codes zu einem neuen Code zusammengeführt, indem sie mehrere Algorithmen durchlaufen, so wird dies im Rahmen der Arbeit als Codeliste bezeichnet. In diesem Zusammenhang wäre auch der phonetische ONCA-Code eine Codeliste, da eine Zeichenkette zunächst in ein Derivat des NYSIIS-Codes codiert und danach für diesen der Soundex-Code ermittelt wird. Codelisten sind zur Erkennung von Unschärfen sinnvoll, indem Werte zunächst mehrfach kodiert und anschließend über ein Proximitätsmaß verglichen werden. Hierzu ein einfaches Beispiel: Eine Zeichenkette wird zunächst buchstabenweise alphabetisch sortiert und in einem zweiten Schritt ein spezifischer Patterncode erzeugt. Dieser Code kann dann als Basis für einen Textvergleich oder zur Partitionierung im Rahmen der Duplikaterkennung dienen.

---

[235] Sparck Jones, K.; Willett, P (Hrsg.).: Readings in Information Retrieval; Morgan Kaufmann Publishers; San Francisco; 1997; S. 306

[236] Vgl. hierzu: Manning, C. D., Schütze, H.: Foundations of Statistical Natural Language Processing; The MIT Press; Cambridge, Massachusetts; 2003; S. 533

[237] Baeza-Yates, R.; Ribeiro-Neto, B.: Modern Information Retrieval; Addison-Wesley; Harlow; 1999; S. 167f

[238] Vgl. hierzu: Witten, I.H.; Moffat, A.; Bell, T.C.: Managing Gigabytes - Compressing and Indexing Documents and Images, 2nd. ed.; Morgan Kaufmann; 1999; San Francisco; S. 149

3.2.2 Proximitätsmaße

3.2.2.1 Grundlagen

Proximitätsmaße vergleichen zwei Werte in Bezug auf ihre Ähnlichkeit. Im Gegensatz zu phonetischen Codes sind Proximitätsmaße eher als domänenunabhängig anzusehen.[239] U. Leser und F. Naumann sehen jedoch auch die Anwendung von Proximitätsmaßen als domänenabhängig an, da bei deren Auswahl durchaus der Inhalt, die Sprache und die Entstehung eines Attributs berücksichtigt werden muss. [240] Diese Sichtweise wird auch von Bilenko u.a. bestätigt, die die Performanz für verschiedene Proximitätsmaße auf verschiedenen Datenbeständen getestet haben.[241] Die Begründungen von U. Leser und F. Naumann sind zwar nachvollziehbar, bei einer gemeinsamen Betrachtung und Einteilung von phonetischen Codes und Proximitätsmaßen, sind die Ersteren allerdings eher der domänenabhängigen und Zweitere der domänenunabhängigen Form zuzuordnen.

3.2.2.2 Numerische Daten

Im Vergleich zu den Proximitätsmaßen für alphanumerische Daten sind die Algorithmen für numerische Daten eher primitiv.[242] Neben einfachen Bereichsabfragen oder der Distanz zwischen zwei Zahlen, können numerische Daten natürlich wie alphanumerische Daten behandelt werden, so dass alle Proximitätsmaße für alphanumerische Daten anwendbar sind, die im nächsten Abschnitt beschrieben werden.

Aus dem Bereich der Cluster- bzw. Segmentierungsverfahren existieren mehrere Proximitätsmaße, wenn für mehr als ein Attribut ein Ähnlichkeits- oder Distanzmaß errechnet werden

---

[239] Monge, A.: An Adaptive and Efficient Algorithm for Detecting Approximately duplicate Database Records, California State University, Long Beach, 2000, S. 3

[240] Werden Begriffe auf Papier über Tastatur in den Computer übertragen, sollte der Abstand von Zeichen auf der Tastatur berücksichtigt werden, um Schreibfehler zu erkennen. Erfolgt die Eingabe aus Grundlage eines Telefongespräches, können eher phonetische Codes sinnvoll sein (vgl. hierzu: Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 334f)

[241] Elmagarmid, A.K.; Ipeirotis, P.G.; Verykios, V.S.: Duplicate Record Detection: A Survey; erschienen in: IEEE Transactions on Knowledge and Data Engineering, Vol. 19, No. 1; IEEE Computer Society; Washington, DC; 2007; S. 5

[242] Elmagarmid, A.K.; Ipeirotis, P.G.; Verykios, V.S.: Duplicate Record Detection: A Survey; erschienen in: IEEE Transactions on Knowledge and Data Engineering, Vol. 19, No. 1; IEEE Computer Society; Washington, DC; 2007; S. 5

soll. Die bekanntesten Distanzmaße hierbei sind die Euklidische Distanz und in verallgemeinerter Form die City-Block-Distanz.[243]

3.2.2.3 Alphanumerische Daten

Zeichenketten können entweder auf ihre exakte Übereinstimmung (Exact String Matching), auf Übereinstimmung von Teilzeichenketten oder aber auf ihre Ähnlichkeit hin untersucht werden. Algorithmen zur Überprüfung der exakten Übereinstimmung[244] haben den Nachteil, dass zwei zu vergleichende Zeichenketten entweder identisch sind oder verschieden voneinander, auch wenn z.B. nur ein Buchstabe in einer der Zeichenketten fehlt. Die Zeichenketten „Cordts“ und „Cords“ stimmen danach nicht überein, obgleich sie offensichtlich das gleiche Objekt beschreiben.

Ein einfacher Ansatz, Unschärfen zwischen Zeichenketten zu berücksichtigen, besteht darin, nur Teilzeichenketten zu vergleichen. Danach würden zwei Zeichenketten übereinstimmen, sofern z.B. die ersten drei Buchstaben übereinstimmen. In diesem Fall würden die Zeichenketten „Cordts“ und „Cords“ als identisch angesehen werden. Das setzt jedoch voraus, dass man Kenntnis darüber besitzt, welcher Teil einer Zeichenkette relativ fehlerfrei ist. Beispielsweise wären die beiden Zeichenketten „Kordts“ und „Cordts“ bei Anwendung dieser Methode nicht identisch. Auch die Verwendung sogenannter Wildcards[245] (Wildcard String Matching) – also Zeichen die in einer Zeichenkette für ein beliebiges Alphabetsymbol in einer Zeichenkette stehen – setzen die Kenntnis darüber voraus, welche Teilzeichenketten weitestgehend fehlerfrei sind. Verwendet man als Wildcardsymbol das Fragezeichen an der ersten zu überprüfenden Stelle, so würden bezogen auf das letzte Beispiel die beiden Zeichenketten übereinstimmen. Die Verwendung des Exact oder Wildcard String Matching ist nur sinnvoll, wenn bestimmte Zeichenfolgen stabil bzw. fehlerfrei bleiben. Ist dies nicht der Fall bzw. hat man keine Kenntnis über die Stabilität von Teilzeichenfolgen, so ist der Einsatz sogenannter Approximate String Matching Algorithmen[246] sinnvoll.

---

[243] Bortz, J.: Statistik für Human- und Sozialwissenschafter; 6. Auflage; Springer Medizin Verlag; Heidelberg; 2005; S. 568ff

[244] eine ausführliche Beschreibung dieser Algorithmen findet sich in Charras, C.; Lecroq, T.: Handbook of Exact String Matching Algorithms, King’s College Publications, o.A., 2004

[245] Schöning, U.: Algorithmik, Spektrum Akademischer Verlag, Heidelberg, 2001, S. 81

[246] Die Bezeichnung Approximate String Matching orientiert sich an Navarro, G.: A Guided Tour to Approximate String Matching; erschienen in: ACM Computing Surveys Vol. 33 Issue 1; ACM Press; New York; 2001, S. 31

Approximate String Matching Algorithmen[247] vergleichen zwei Zeichenketten in Bezug auf ihre Ähnlichkeit bzw. auf ihre Unterschiede. Entsprechend unterscheidet man zwischen Distanzmaßen, deren Werte die Unterschiede zwischen zwei Zeichenketten quantifizieren und Ähnlichkeitsmaßen, die die Ähnlichkeit normalerweise als Werte zwischen 0 und 1 angeben.

Distanzmaße können in ein Ähnlichkeitsmaß umgewandelt werden, indem das Distanzmaß z.B. durch die Länge der breiteren Zeichenkette dividiert und von 1 abgezogen wird:

$$\ddot{A}hnlichkeitsma\ss(s1, s2) = 1.0 - \frac{Dist.ma\ss(s1, s2)}{\max(len(s1), len(s2)}$$

Ursprünglich stammen Algorithmen zum Approximate String Matching aus der Bioinformatik-Forschung, um zu überprüfen, ob zwei Abschnitte eines Genoms Ähnlichkeiten aufweisen.[248] Sie haben sich aber auch in anderen Bereichen wie z.B. dem Information Retrieval als sehr wichtig erwiesen.

Daneben kann außerdem zwischen zeichen-, tokenbasierten und hybriden Algorithmen unterschieden werden. Zeichenbasierte Algorithmen berechnen das Proximitätsmaß zwischen zwei Zeichenketten, indem die beiden Zeichenketten zeichenweise verglichen werden. Da Zeichenketten jedoch auch aus mehreren Werten bestehen können (z.B. kann ein Attribut „Adresse“ mehrere Werte wie Vor-, Nachname, Strasse usw. enthalten), existieren tokenbasierte Verfahren, die die zu vergleichenden Zeichenketten in einzelne Token[249] zerlegen und anschließend die Ähnlichkeit anhand der einzelnen Token ermitteln. Eine Kombination aus zeichen- und tokenbasierten Verfahren sind die hybriden Algorithmen, die Zeichenketten in einzelne Token aufteilen und diese anschließend mit zeichenbasierten Algorithmen vergleichen.

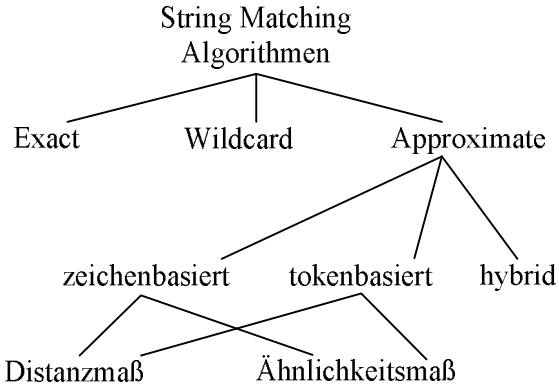
---

[247] auch als “string matching allowing errors” bezeichnet, vgl. hierzu: Navarro, G.; Raffinot, M.: Flexible Pattern Matching in Strings; Cambridge University Press; Cambridge; 2002; S. 145 oder Baeza-Yates, R.; Ribeiro-Neto, B.: Modern Information Retrieval; Addison-Wesley; Harlow; 1999; S. 216

[248] Heller, M.: Approximative Indexierungstechniken für historische deutsche Textvarianten; erschienen in: Historical Social Research / Historische Sozialforschung, Vol. 31 (2006), No. 3; Zentrum für Historische Sozialforschung e.V.; Köln; 2006; S. 292

[249] Token können sowohl einzelne Werte sein, die z.B. durch ein Komma oder ein Leerzeichen getrennt sind, aber auch die Aufteilung in gleich große Teilzeichenketten kann sinnvoll sein (siehe hierzu 3.2.1.8 Tokenizer). Nach C. Fox sind Tokens allgemein Gruppen von Zeichen mit gemeinschaftlicher Bedeutung (Fox, C.: Lexical Analysis and Stoplists; erschienen in: Frakes, W.B.; Baeza-Yates, R. (Hrsg.): Information Retrieval - Data Structures & Algorithms; Prentice-Hall; Upper Saddle River, New Jersey; 1992; S. 102)

Abb. 18: Proximitätsmaße alphanumerische Daten



Quelle: eigene Darstellung

Im Folgenden werden die bekanntesten Approximate String Matching Algorithmen beschrieben.

Zu den bekanntesten zeichenbasierten Distanzmaßen gehören:

- Hamming,
- Levenshtein,
- Smith-Waterman,
- Needleman-Wunsch,
- Longest Common Subsequence.

Wie bereits oben erwähnt, ist das Distanzmaß umso größer, je mehr sich zwei Zeichenketten voneinander unterscheiden. Das einfachste Distanzmaß zur Berechnung der Unterscheidung ist die Hamming-Distanz. Dieses Distanzmaß zählt die Anzahl von unterschiedlichen Zeichen an der gleichen Position von zwei Wörtern. Um eine Zeichenkette in die andere zu konvertieren, verwendet die Hamming-Distanz als Operatoren also nur das Vertauschen. Daher ist das Verfahren nur für Zeichenketten gleicher Länge geeignet[250] und kann z.B. benutzt werden, um den Abstand von Zahlen fixer Länge zu berechnen.[251] Ursprünglich ist die Anfang der 50er Jahre entwickelte Hamming-Distanz zur Fehlererkennung bei der

[250] Merkl, R.; Waack, S.: Bioinformatik interaktiv; Wiley-VCH; Weinheim; 2003; S. 92
[251] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 105

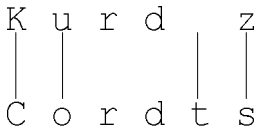
Übertragung binärer Daten entwickelt worden.[252] Die Buchstaben A und C binär als ASCII-Codes betrachtet, haben danach einen Abstand von 1:

0100 00**0**1 (A)
0100 00**1**1 (C)

Entsprechend kann die Hamming-Distanz natürlich auch auf beliebige Zeichenketten gleicher Länge angewendet werden, wie z.B. zum Vergleich von Postleitzahlen (25**7**5**6** und 25**8**7**6** haben danach eine Hamming-Distanz von 2).[253]

Das bekannteste Distanzmaß wurde von dem russischen Mathematiker V. Levenshtein und F. Damerau entwickelt und wird als Levenshtein- oder Damerau-Levenshtein-Distanz bezeichnet.[254] Im Gegensatz zur Hamming-Distanz werden hier bei der Berechnung nicht nur Vertauschungen berücksichtigt. Vielmehr wird die geringste Anzahl an Zeichenoperationen (Einfügen, Löschen, Austausch, Kopieren) berechnet, die notwendig ist, um eine Zeichenkette in die andere zu transformieren. Dabei werden für die drei Operationen Einfügen, Löschen und Austausch Kosten von einer Einheit berechnet. So beträgt z.B. zwischen den beiden Nachnamen „Cordts“ und „Kurdz“ die Levenshtein-Distanz 4, da drei Buchstaben getauscht werden müssen (‚K **à** C’, ‚u **à** o’, ‚z **à** s’) und ein Buchstabe hinzugefügt wird (‚t’):

Abb. 19: Beispiel zur Levenshtein-Distanz



Quelle: eigene Darstellung

[252] Heller, M.: Approximative Indexierungstechniken für historische deutsche Textvarianten; erschienen in: Historical Social Research / Historische Sozialforschung, Vol. 31 (2006), No. 3; Zentrum für Historische Sozialforschung e.V.; Köln; 2006; S. 293

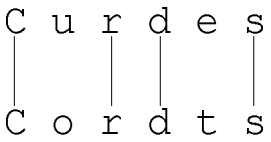
[253] Ein ähnliches Maß wie die Hamming-Distanz ist das Typewriter-Distanzmaß. Hierbei werden auch alle unterschiedlichen Zeichen gemessen. Es wird jedoch nicht pro unterschiedlichem Zeichen eine Distanz von 1 vergeben, sondern es wird errechnet, wie weit die beiden unterschiedlichen Zeichen auf einer Tastatur voneinander entfernt sind (Beispiel: zwischen den Tasten ‚Q’ und ‚R’ liegen zwei Tasten, so dass der Abstand dieser beiden Buchstaben 2 beträgt)

[254] Vgl. hierzu: Heller, M.: Approximative Indexierungstechniken für historische deutsche Textvarianten; erschienen in: Historical Social Research / Historische Sozialforschung, Vol. 31 (2006), No. 3; Zentrum für Historische Sozialforschung e.V.; Köln; 2006; S. 293

Die Smith-Waterman- und auch die Needleman-Wunsch-Distanz sind Erweiterungen der Levenshtein-Distanz. Die Needleman-Wunsch-Distanz ordnet den einzelnen Operationen unterschiedliche Kosten zu. Das Verfahren berechnet dann die geringsten Kosten, um eine Zeichenkette in eine andere umzuwandeln. Die Kosten für das Einfügen, Ändern oder Löschen eines Zeichens sind danach Parameter des Algorithmus.[255]

Indem nur die Operationen Löschen mit Kosten von einer Einheit erlaubt werden[256], kann die längste gemeinsame Teilzeichenfolge (Longest Common Subsequence) berechnet werden, um die Ähnlichkeit zwischen zwei Zeichenketten auszudrücken[257]. Einer der bekanntesten Algorithmen zur Berechnung der „Longest Common Subsequence“ ist der Hirschberg-Algorithmus[258]. Die längste gemeinsame Teilzeichenfolge der beiden Zeichenketten „Cordts“ und „Curdes“ beträgt demnach 4.[259]

Abb. 20: Beispiel zur Longest Common Subsequence



Quelle: eigene Darstellung

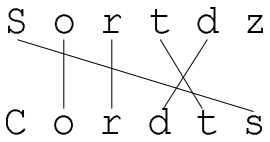
Zu den bekanntesten zeichenbasierten Ähnlichkeitsmaßen gehören:

- Jaro,
- Jaro-Winkler,
- Lynch-Jaro-Winkler,
- Ratcliff-Obershelp.

[255] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 105
[256] Baeza-Yates, R.; Ribeiro-Neto, B.: Modern Information Retrieval; Addison-Wesley; Harlow; 1999; S. 148
[257] Durch die Longest Common Subsequence wird eigentlich die Ähnlichkeit zwischen zwei Zeichenketten ausgedrückt. Da das Verfahren jedoch an die Algorithmen von Distanzmaßen angelehnt ist, spricht man meistens von Distanzmaß.
[258] Vgl. hierzu: Hirschberg, D. S.: A linear space algorithm for computing maximal common subsequences; erschienen in: Communications of the ACM; Vol. 18, No. 6; ACM Press, New York; 1975; S. 341-343
Stephen, G.A.: String Searching Algorithms, Lectures Notes Series on Computing - Vol. 3; World Scientific; Singapore; 2000; S. 57ff
[259] Normalisiert auf die längste Zeichenkette ergibt sich ein Ähnlichkeitsmaß von: 4 / 6 = 66,67%

Ähnlichkeitsmaße sind umso größer, je ähnlicher zwei zu vergleichende Zeichenketten sind. Am häufigsten in der Literatur wird das Jaro-Ähnlichkeitsmaß erwähnt, das ursprünglich zum Vergleich von Vor- und Nachnamen entwickelt wurde.[260] Es berücksichtigt die Anzahl und die Reihenfolge übereinstimmender Zeichen. Zwei übereinstimmende Zeichen müssen dabei nicht an der gleichen Position erscheinen, sondern können sich überkreuzen. Das Jaro-Ähnlichkeitsmaß berücksichtigt also Einfügungen, Löschungen und Transpositionen. Das in Abb. 21 dargestellte Beispiel vergleicht die beiden Zeichenketten „Cordts“ und „Sortdz“. Dazu werden zunächst die übereinstimmenden Zeichen beider Zeichenketten ermittelt. Ein übereinstimmendes Zeichen muss sich dabei innerhalb der Hälfte der Länge der kürzeren Zeichenkette befinden. Im Beispiel ist die minimale Zeichenkettenlänge 6. Da der Abstand des Zeichens ‚S’ in der ersten Zeichenkette größer als 6 / 2 = 3 ist, wird das Zeichen ‚S’ bei der Berechnung der übereinstimmenden Zeichen nicht berücksichtigt. Damit ergeben sich vier übereinstimmende Zeichen (‚o’, ‚r’, ‚t’, ‚d’). ‚t’ und ‚d’ sind vertauscht, so dass sich eine Überkreuzung ergibt.[261]

Abb. 21: Beispiel zum Jaro-Maß



Quelle: eigene Darstellung

Das Jaro-Ähnlichkeitsmaß errechnet sich wie folgt:[262]

$$sim_{jaro}(s1, s2) = \frac{1}{3} \cdot \left(\frac{common}{s1\_len} + \frac{common}{s2\_len} + \frac{(common - trans}{common} \right)$$

mit

s1_len	Länge der ersten Zeichenkette
s2_len	Länge der zweiten Zeichenkette
common	Anzahl übereinstimmender Zeichen
trans	Anzahl Überkreuzungen

[260] Elmagarmid, A.K.; Ipeirotis, P.G.; Verykios, V.S.: Duplicate Record Detection: A Survey; erschienen in: IEEE Transactions on Knowledge and Data Engineering, Vol. 19, No. 1; IEEE Computer Society; Washington, DC; 2007; S. 3

[261] Vgl. hierzu: Porter, E. H.; Winkler, W. E.: Approximate String Comparison and its Effect on an Advanced Record Linkage System; U.S. Bureau of the Census, Research Report; Washington; 1997; S. 2

[262] Vgl. hierzu: Cohen, W. u.a.: A Comparison of String Distance Metrics for Name-Matching Tasks; America Assocation for Artificial Intelligence; Menlo Park; 2003; S. 2

Damit ergibt sich für das Beispiel eine Ähnlichkeit von:

$$sim_{jaro}(s1, s2) = \frac{1}{3} \cdot \left(\frac{4}{6} + \frac{4}{6} + \frac{(4-1)}{4} \right) = 69{,}44\%$$

Eine Erweiterung des Jaro-Ähnlichkeitsmaßes ist der Jaro-Winkler-Algorithmus, der zunächst das Jaro-Maß berechnet und anschließend das Ähnlichkeitsmaß abhängig von der Anzahl der am Anfang übereinstimmenden Zeichen als Korrekturfaktor erhöht. Das Jaro-Winkler-Maß basiert auf einer Untersuchung von J. Pollock und A. Zamorra (1984), in der nachgewiesen wurde, dass am Anfang der Zeichenkette die wenigsten Fehler auftreten und die Fehlerrate zum Ende der Zeichenkette hin gleichmäßig ansteigt.[263] Die Formel zur Berechnung des Jaro-Winkler-Maßes sieht wie folgt aus:

$$sim_{jaro-winkler}(s1, s2) = sim_{jaro} + \frac{commonprefix}{10} \cdot (1 - sim_{jaro})$$

mit

commonprefix	Anzahl am Anfang übereinstimmender Zeichen, wobei maximal 4 Zeichen berücksichtigt werden

Vergleicht man die beiden Zeichenketten „Cordts“ und „Cortdz“, so ergibt sich als Jaro-Maß eine Übereinstimmung von 75% und als Jaro-Winkler-Maß eine Übereinstimmung von 82,50%:

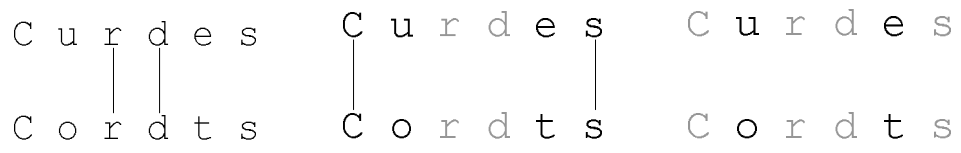
$$sim_{jaro-winkler}(s1, s2) = 75\% + \frac{3}{10} \cdot (1 - 75\%) = 82{,}50\%$$

1994 wurde der Jaro-Winkler-Algorithmus von Lynch und Winkler[264] auf Basis detaillierter Vergleiche unterschiedlicher Algorithmusvarianten mit Personendaten erweitert.[265]

[263] Porter, E. H.; Winkler, W. E.: Approximate String Comparison and its Effect on an Advanced Record Linkage System; U.S. Bureau of the Census, Research Report; Washington; 1997; S. 3
[264] Zur Implementierung siehe: http://www.census.gov/geo/msb/stand/strcmp.c
[265] Porter, E. H.; Winkler, W. E.: Approximate String Comparison and its Effect on an Advanced Record Linkage System; U.S. Bureau of the Census, Research Report; Washington; 1997; S. 3f

Der Ratcliff-Obershelp-Algorithmus von 1983 wurde entwickelt, um in Lernsoftware richtige Lösungen zu erkennen, die als Textlösungen eingegeben werden und Tippfehler enthalten können. Dazu ermittelt der Algorithmus zunächst den „Longest Common Substring“, die längste zusammenhängende Teilzeichenkette.[266] Danach werden rekursiv die verbleibenden Teilzeichenketten links und rechts von der gefundenen erneut nach der längsten zusammenhängenden Teilzeichenkette untersucht. Die Anzahl der Zeichen der gefundenen Teilzeichenketten werden summiert und durch die Länge der beiden Zeichenketten dividiert. Das Ratcliff-Obershelp-Maß der beiden Zeichenketten „Cordts“ und „Curdes“ beträgt danach 66,67%.[267]

Abb. 22: Beispiel zum Ratcliff-Obershelp-Maß



Quelle: eigene Darstellung

Im Beispiel (siehe Abb. 22) besteht die erste längste übereinstimmende Zeichenkette, aus den Buchstaben ‚rd’ (insgesamt 4 gleiche Zeichen). Links und rechts davon ergeben sich insgesamt jeweils 2 Zeichen (links ‚C’ und rechts ‚s’). Damit errechnet sich das Maß wie folgt:

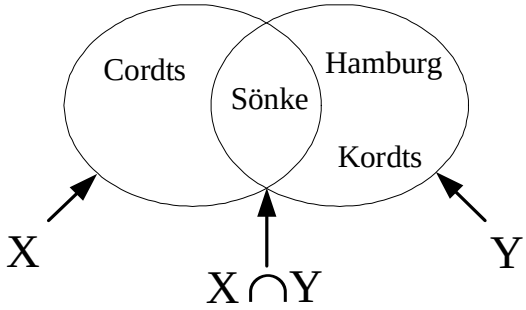
$$sim_{Ratcliff-Obershelp}(s1, s2) = \frac{4+2+2}{6+6} = 66{,}67\%$$

Zwei unterschiedliche Zeichenketten können nicht nur einen atomaren Wert enthalten, sondern auch aus mehreren Werten bestehen, die durch Trennzeichen wie Komma, Leerzeichen usw. getrennt sind. Die bisher beschriebenen Algorithmen vergleichen zwei Zeichenketten zeichenweise. Solche Algorithmen sind sinnvoll für kurze Zeichenketten, die aus einzelnen atomaren Werten bestehen. Vergleicht man dagegen aus mehreren Einzelworten bestehende Zeichenketten, liefern tokenbasierte Verfahren effizientere Ergebnisse. So ergibt z.B. ein Vergleich der beiden Zeichenketten „Cordts, Sönke“ und „Sönke Cordts“ mit den

[266] Hierbei sind Lücken nicht erlaubt, vergleiche hierzu „Longest Common Subsequence“
[267] Vgl. hierzu: Ratcliff, J.W.; Metzener, D.E.: Pattern Matching: The Gestalt Approach; erschienen in: Dr. Dobb's Journal, No. 141; Boulder, Colorado; 1988; S. 46ff

beschriebenen zeichenbasierten Verfahren (ausgenommen das Hirschberg-Verfahren) eine Ähnlichkeit von 0, obwohl intuitiv beide Werte identisch sind. Einfache tokenbasierte Verfahren sind auf Mengenoperationen zurückzuführen. Hierbei wird normalerweise die Anzahl übereinstimmender Wörter zwischen den beiden Zeichenketten in Bezug zu einer anderen Maßzahl – z.B. der Anzahl aller Wörter in beiden Zeichenketten – gesetzt. Abbildung 23 gibt einen Überblick zu den Kombinationsmöglichkeiten.

Abb. 23: Mengenoperationen auf Zeichenketten



Quelle: eigene grafische Darstellung, in Anlehnung an: Michel, C.: Cardinal, nominal or ordinal similarity measures in comparative evaluation of information retrieval process; erschienen in: Actes du congrès LREC 2000 Second international conference on language ressource and evaluation; Athènes, 2000; S. 1510

Zu den bekanntesten tokenbasierten Distanz- und Ähnlichkeitsmaßen gehören

- Matching Coefficient,
- Dice,
- Jaccard- oder Tanimoto,
- Overlap,
- Kosinus,
- City-Block- oder L1,
- Euklidisches Distanz- oder L2.

Daneben existieren tokenbasierte Verfahren, die auf statistischer Wahrscheinlichkeitsrechnung basieren. Zu den bekanntesten Verfahren hierzu gehören

- TF-IDF und
- Fellegi und Sunter

Grundsätzlich sind alle hier genannten tokenbasierten Verfahren auch zur Duplikaterkennung geeignet. Dieses gilt jedoch insbesondere für die beiden letztgenannten probabilistischen Verfahren. Gerade das Fellegi und Sunter-Maß wurde ursprünglich zur Duplikaterkennung entwickelt und wird deshalb auch erst im Abschnitt zu diesem Thema (siehe Abschnitt 3.5.3) ausführlich behandelt.

Der Matching Coefficient ist der einfachste Algorithmus und zählt einfach alle exakt übereinstimmenden Zeichenketten. Für das Beispiel der beiden Zeichenketten „Sönke Cordts“ und „Sönke Kordts Hamburg“ würde sich der Wert 1 ergeben.[268]

Das Dice-Maß ergibt sich, indem die Anzahl übereinstimmender Wörter mit 2 multipliziert und durch die Anzahl aller Wörter dividiert wird:[269]

$$sim_{Dice} = \frac{2|X \cap Y|}{|X| + |Y|}$$

Das Dice-Maß für die beiden Zeichenketten „Sönke Cordts“ und „Sönke Kordts Hamburg“ beträgt danach

$$sim_{Dice} = \frac{2 \cdot 1}{2 + 3} = 40{,}0\%$$

Das Jaccard- oder Tanimoto-Maß[270] setzt dagegen die Anzahl übereinstimmender Wörter ins Verhältnis zur Anzahl aller unterschiedlichen Wörter:[271]

$$sim_{Jaccard} = \frac{|X \cap Y|}{|X \cup Y|}$$

[268] Zur Normalisierung im entwickelten Framework wird die Zeichenkette mit den meisten Wörtern ermittelt und der Matching Coefficient durch diesen Wert dividiert

[269] Manning, C. D., Schütze, H.: Foundations of Statistical Natural Language Processing; The MIT Press; Cambridge, Massachusetts; 2003; S. 299

[270] Dieses Maß von Jaccard (1908) und Rogers/Tanimoto wird auch als S-Koeffizient bezeichnet (vgl. hierzu: Bortz, J.: Statistik für Human- und Sozialwissenschafter; 6. Auflage; Springer Medizin Verlag; Heidelberg; 2005; S. 567)

[271] Bilenko, M.: Learnable Similarity Functions and Their Applications to Record Linkage and Clustering – Doctoral Dissertation Proposal, University of Texas, Austin, 2003, S. 8

Damit ergibt sich für das eben genannte Beispiel ein Jaccard-Maß von 25,0%, da ein gemeinsames Wort und vier voneinander verschiedene Wörter vorkommen:

$$sim_{Jaccard} = \frac{1}{4} = 25{,}0\%$$

Eine geringe Anzahl von gemeinsamen Wörtern wird hier im Gegensatz zum Dice-Maß eher berücksichtigt. Eine geringe Anzahl an Überschneidungen ergibt einen geringeren Jaccard-Wert (siehe Beispielwerte).[272]

Das Overlap-Maß setzt die Anzahl übereinstimmender Worte ins Verhältnis zu der Zeichenkette mit der geringsten Anzahl an Wörtern. Sind also die Wörter einer der beiden Zeichenketten eine Teilmenge der anderen Zeichenkette, so ergibt sich beim Overlap-Maß immer eine vollständige Übereinstimmung.[273] Die Formel lautet:

$$sim_{Overlap} = \frac{|X \cap Y|}{\min(|X|, |Y|)}$$

Für das obige Beispiel ergibt sich damit ein 50%ige Übereinstimmung der beiden Zeichenketten:

$$sim_{Overlap} = \frac{1}{\min(2,3)} = 50\%$$

Das Kosinus-Maß ergibt für zwei Zeichenketten, die aus der gleichen Anzahl an Wörtern bestehen, das gleiche Ergebnis wie das Dice-Maß. Bei einer unterschiedlichen Anzahl an Wörtern ergibt das Kosinus-Maß eine höhere Übereinstimmung als das Dice-Maß. Die Formel für das Kosinus-Maß sieht folgendermaßen aus:

$$sim_{Cosine} = \frac{|X \cap Y|}{\sqrt{|X| \cdot |Y|}}$$

[272] Manning, C. D., Schütze, H.: Foundations of Statistical Natural Language Processing; The MIT Press; Cambridge, Massachusetts; 2003; S. 299

[273] Vgl. hierzu: Manning, C. D., Schütze, H.: Foundations of Statistical Natural Language Processing; The MIT Press; Cambridge, Massachusetts; 2003; S. 299

Vergleicht man beispielsweise zwei Zeichenketten, bei der die erste aus 1000 und die zweite aus einem Wort besteht, das in der ersten Zeichenkette vorkommt, ergibt sich für das Dice-Maß ein Wert von etwa 0,2%, für das Kosinus-Maß dagegen von 3%.[274] Für das obige Beispiel lässt sich eine Übereinstimmung von 40,8% errechnen:

$$sim_{Cosine} = \frac{1}{\sqrt{2 \cdot 3}} = 40{,}8\%$$

Bei den bisher beschriebenen tokenbasierten Verfahren handelt es sich um Algorithmen, die binäre Vektoren verwenden. Das Kosinus-Maß z.B. kann aber auch als Ähnlichkeitsmaß verwendet werden, wenn es sich um reellwertige Vektoren handelt.[275]

Neben diesen Maßen, die für binäre Attribute geeignet sind, können auch Distanzmaße für metrische Attribute verwendet werden. Die bekanntesten sind die City-Block-Distanz und die euklidische Distanz. Die City-Block-Distanz berechnet die Summe aller absoluten Abstände zwischen zwei zu vergleichenden Zahlen. Bei zwei zu vergleichenden Zeichenketten soll die Distanz für ein Wort 1 sein, wenn ein Wort nur in einer der beiden Zeichenketten und 0, wenn das Wort in beiden Zeichenketten vorkommt:

$$dist_{CityBlock} = \sum_{j=1}^{n} |x_i - y_i|$$

Als City-Block-Distanz[276] für das obige Beispiel ergibt sich damit ein Abstand von 2. Bei der euklidischen Distanz werden zusätzlich die einzelnen Abstände quadriert und zum Schluss die Quadratwurzel gezogen, so dass sich für das Beispiel eine Distanz von 1,41 ergibt.[277]

$$dist_{euklidischeDis\tan z} = \sqrt{\sum_{j=1}^{n} (x_i - y_i)^2}$$

[274] Vgl. hierzu: Manning, C. D., Schütze, H.: Foundations of Statistical Natural Language Processing; The MIT Press; Cambridge, Massachusetts; 2003; S. 300
[275] siehe weiter unten zum TF-IDF-Maß
[276] Die Bezeichnung City-Block-Distanz wurde übrigens im Gegensatz zur euklidischen Distanz (diese gibt sozusagen die direkte „Luftlinie“ zwischen zwei Punkten) verwendet, da sie die Entfernung angibt, die z.B. ein Auto in einer Stadt mit rechtwinklig verlaufenden Strassen zurücklegen muss (vgl. hierzu: Bortz, J.: Statistik für Human- und Sozialwissenschafter; 6. Auflage; Springer Medizin Verlag; Heidelberg; 2005; S. 570)
[277] Vgl. hierzu: Backhaus, K. u.a.: Multivariate Analysemethoden; 10. Auflage; Springer Verlag; Berlin u.a.; 2003; S. 484ff

TF-IDF-Verfahren (Term Frequency – Inverse Document Frequency) wurden im Bereich des Information Retrieval entwickelt, um ähnliche Zeichenketten in Dokumenten zu finden.[278] Dabei wird den einzelnen Wörtern abhängig von ihrer Häufigkeit insgesamt und in der eigenen Zeichenkette ein Gewicht zugeteilt.[279] Die Idee des TF-IDF-Verfahrens basiert auf einem Artikel von K. Sparck-Jones aus dem Jahr 1972[280] und besteht aus zwei Grundüberlegungen: Auf der einen Seite muss ermittelt werden, welche Wörter in einer Zeichenkette diese am besten beschreiben. Auf der anderen Seite muss bestimmt werden, welche Wörter in der Zeichenkette sich von den Wörtern aller anderen Zeichenketten besser unterscheiden.[281] Ein Wort, das in sehr vielen Zeichenketten vorkommt, ist als Diskriminator zwischen den Zeichenketten nicht geeignet.[282] Dabei sollte das entstehende Maß eine Balance dieser beiden Grundüberlegungen darstellen. Dazu wird für jedes Wort in einer Zeichenkette ein Gewicht berechnet, das von seiner Häufigkeit in der eigenen Zeichenkette (Term-Frequency) und in allen betrachteten Zeichenketten (Document-Frequency) abhängt.[283] Wörter die in der eigenen Zeichenkette oft vorkommen, erhalten ein hohes und Wörter, die in fast jeder der betrachteten Zeichenketten vorkommen, ein niedriges Gewicht.[284] Die Unterscheidbarkeit zwischen den einzelnen Zeichenketten wird durch die Inverse der Häufigkeit eines Wortes in allen Zeichenketten gemessen (IDF, Inverse-Document-Frequency).[285]

Als Term-Frequency kann in der einfachsten Variation die Häufigkeit des auftretenden Wortes in der zu untersuchenden Zeichenkette verwendet werden. Daneben existieren verschiedene Variationen zur Berechnung der Term-Frequency. So kann z.B. die Anzahl jedes Wortes durch die Anzahl des am häufigsten in der Zeichenkette vorkommenden Wortes

[278] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 106
[279] Vgl. Sparck-Jones, K.; Willett, P (Hrsg.).: Readings in Information Retrieval; Morgan Kaufmann Publishers; San Francisco; 1997; S. 306
[280] Sparck-Jones, K.: A statistical interpretation of term specificity and its application in retrieval; erschienen in: Journal of Documentation, Vol. 28, No. 1; Emerald Group; Bradford, United Kingdom; 1972; S. 11-21
[281] Vgl. hierzu: Baeza-Yates, R.; Ribeiro-Neto, B.: Modern Information Retrieval; Addison-Wesley; Harlow; 1999; S. 29
[282] Vgl. hierzu: Robertson, S.: Understanding Inverse Document Frequency: On theoretical arguments for IDF; erschienen in: Journal of Documentation, Vol. 60, No. 5; Emerald Group; Bradford, United Kingdom; 2004; S. 503
[283] Im Information Retrieval geht es um die Suche von Dokumenten nach bestimmten Suchbegriffen. Im Rahmen dieser Arbeit wird ein Dokument jedoch als Zeichenkette bestehend aus mehreren Wörtern betrachtet. Insofern sind die Begriffe Dokument und Zeichenkette in diesem Zusammenhang als synonym anzusehen.
[284] Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 339f
[285] In diesem Zusammenhang wird auch von globalen kontextunabhängigen (Document Frequency) und lokalen kontextabhängigen Einflussfaktoren gesprochen (vgl. hierzu: Ferber, R.: Information Retrieval; dpunkt Verlag; Heidelberg; 2003; S. 66f)

dividiert und zusätzlich von diesem Wert noch einmal der Logarithmus gebildet wird (vor allem bei längeren Texten).[286]

Die IDF lässt sich berechnen als Logarithmus der Anzahl aller Zeichenketten dividiert durch die Anzahl der Zeichenketten, in der das Wort vorkommt:

$$idf_i = \log_b \frac{N}{n_i}$$

Das Gewicht für ein Wort ist nun das Produkt aus der Term-Frequency und der Inverse-Document-Frequency: [287]

$$w_{i,j} = tf_{i,j} \cdot idf_i$$ [288]

Als Beispiel zur Berechnung der Gewichte sollen die folgenden drei Datensätze aus Tabelle 11 betrachtet werden:

Tab. 11: Beispieldaten zur Berechnung der TF-IDF-Maße

Nr.	Vorname	Nachname	Ort
1	Rudolf	Muster	Muster
2	Hans	Muster	Muster
3	Hans	Müller	Hamburg

Quelle: eigene Darstellung

Aus den Beispieldaten lassen sich folgende Häufigkeiten und Gewichte für die einzelnen Wörter errechnen:

[286] Vgl. hierzu: Manning, C. D., Schütze, H.: Foundations of Statistical Natural Language Processing; The MIT Press; Cambridge, Massachusetts; 2003; S. 544

[287] Vgl. hierzu: Baeza-Yates, R.; Ribeiro-Neto, B.: Modern Information Retrieval; Addison-Wesley; Harlow; 1999; S. 29f

[288] TF-IDF ist letztendlich eine Sammlung von Rankingformeln nach dem gleichen hier beschriebenen Grundprinzip. Daher gibt es bei den Formeln zur Berechnung des Gewichts unterschiedliche Varianten. Die hier verwendete Formel stammt von Salton, 1989 und wird in der Literatur am häufigsten erwähnt (vgl. hierzu: Dörre, J.; Gerstl, P.; Seiffert, R.: Volltextsuche und Text Mining; erschienen in: Carstensen, K.-U. u.a. (Hrsg.): Computerlinguistik und Sprachtechnologie, 2. Auflage; Spektrum Akademischer Verlag; Heidelberg; 2004; S.487)

Tab. 12: Berechnung der TF-IDF-Maße

Wörter / Datensatz	Rudolf	Muster	Hans	Müller	Hamburg
Nr. 1	1	2	0	0	0
Nr. 2	0	2	1	0	0
Nr. 3	0	0	1	1	1

Datensatz	Wort	$tf_{i,j}$	idf_i	Gewichte $w_{i,j} = tf_{i,j} \cdot idf_i$
Nr. 1	Rudolf	1	$ln(3/1) = 1{,}10$	1,10
	Muster	2	$ln(3/2) = 0{,}41$	0,81
Nr. 2	Muster	2	$ln(3/2) = 0{,}41$	0,81
	Hans	1	$ln(3/2) = 0{,}41$	0,41
Nr. 3	Hans	1	$ln(3/2) = 0{,}41$	0,41
	Müller	1	$ln(3/1) = 1{,}10$	1,10
	Hamburg	1	$ln(3/1) = 1{,}10$	1,10

Wort / Datensatz	Hans	Muster	Müller	Hamburg	$\sqrt{\sum_{j=1}^{n} w1_{i,j}^{\,2}}$
Nr. 2	0,41	0,81	0	0	0,91
Nr. 3	0,41	0	1,10	1,10	1,61
$w1_{i,j} w2_{i,j}$	0,16	0	0	0	

$$sim(Nr2, Nr3)_{tfidf} = \frac{0{,}16}{0{,}91 \cdot 1{,}61} = 0{,}11$$

Quelle: eigene Darstellung

Die Ähnlichkeit zwischen zwei Zeichenketten wird nun ermittelt, indem für jede Zeichenkette die Gewichte der einzelnen Worte und anschließend der Kosinus dieser beiden Gewichtsvektoren berechnet wird:[289]

$$sim_{tfudf} = \frac{\sum_{j=1}^{n} w1_{i,j} w2_{i,j}}{\sqrt{\sum_{j=1}^{n} w1_{i,j}^{\,2}} \sqrt{\sum_{j=1}^{n} w2_{i,j}^{\,2}}}$$

[289] Vgl. hierzu: Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 340

Danach beträgt die Ähnlichkeit zwischen den Datensätzen 1 und 2 etwa 53,1% und für die beiden Datensätze 2 und 3 etwa 11,3%. Für die Datensätze 1 und 3 beträgt die Ähnlichkeit 0%, da kein Wort übereinstimmt:

$$sim(Nr1, Nr2)_{tfidf} = \frac{0{,}81 \cdot 0{,}81}{\sqrt{1{,}10^2 + 0{,}81^2} \cdot \sqrt{0{,}81^2 + 0{,}41^2}} = \frac{0{,}66}{1{,}37 \cdot 0{,}91} = 0{,}53$$

$$sim(Nr2, Nr3)_{tfidf} = \frac{0{,}41 \cdot 0{,}41}{\sqrt{0{,}41^2 + 0{,}81^2} \cdot \sqrt{0{,}41^2 + 1{,}10^2 + 1{,}10^2}} = \frac{0{,}16}{0{,}91 \cdot 1{,}61} = 0{,}11$$

Tokenbasierte String Matching Algorithmen haben im Gegensatz zu den zeichenbasierten den Nachteil, dass sie Unschärfen in den einzelnen Wörtern nicht berücksichtigen. Die beiden Zeichenketten „Sönke Cordts“ und „Kordts, Sönk“ würden bei den beschriebenen zeichenbasierten Algorithmen 0% bis maximal 50% Ähnlichkeit ergeben, bei den tokenbasierten dagegen immer 0% zurückliefern. Obwohl sich beide Zeichenketten stark ähneln, wird dies durch die bisher beschriebenen Verfahren nicht widergespiegelt.

Eine erste Problemlösung bietet die Verwendung sogenannter N-Gram-Tokenizer (siehe hierzu Abschnitt 3.2.3 Tokenizer). Dabei wird die Zeichenkette nicht wie bisher in einzelne Wörter, sondern in gleich große Teilzeichenketten aufgeteilt. Diese werden dann verwendet, um die tokenbasierten Maße zu berechnen. Teilt man nun die obigen Zeichenketten in Teilzeichenketten der Größe 1 auf (als 1-Gram bezeichnet), so ergibt sich eine Übereinstimmung bei den tokenbasierten Maßen von durchschnittlich 81,3%.

Die zweite Möglichkeit der Problemlösung besteht in der Kombination von tokenbasierten mit zeichenbasierten Algorithmen. Bei den tokenbasierten Verfahren werden die einzelnen Wörter auf exakte Gleichheit überprüft. Stattdessen ist es natürlich möglich, die einzelnen Wörter mit zeichenbasierten Verfahren zu vergleichen. Ein solches hybrides String Matching Verfahren stammt von A.E. Monge und C.P. Elkan und wird als rekursiver Field Matching Algorithmus bezeichnet.[290] Dabei wird jedes Wort aus der ersten Zeichenkette mit jedem Wort der zweiten Zeichenkette über einen zeichenbasierten String Matching Algorithmus

[290] Monge, A. E.; Elkan, C. P.: The field matching problem: Algorithms and applications; erschienen in: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining; Portland, Oregon; 1996; S. 268

verglichen und die maximale Maßzahl verwendet. Alle so errechneten Maßzahlen werden summiert und durch die Anzahl der Wörter in der ersten Zeichenkette dividiert:[291]

$$sim_{MongeElkan} = \frac{1}{|X|} \sum_{i=1}^{|X|} \max_{j=1}^{|Y|} sim_{charbased}(X_i, Y_j)$$

Für das obige Beispiel ergibt sich unter Verwendung der beschriebenen zeichenbasierten Verfahren eine durchschnittliche Monge-Elkan-Maßzahl von 87,6%.[292]

Ming Li u.a.[293] messen die Ähnlichkeit von zwei Zeichenketten, indem diese komprimiert werden und auf Grundlage der komprimierten Daten eine Maßzahl berechnet wird. Bei der Komprimierung von Daten geht es um die Verringerung der Datenmenge. Dabei unterscheidet man zwischen verlustloser und verlustbehafteter Komprimierung. Bei der verlustbehafteten Komprimierung (lossy compression) gehen im Gegensatz zur verlustlosen Komprimierung (lossless compression) Daten der ursprünglichen Datenmenge verloren, die allerdings für den Anwender nicht unmittelbar erkennbar sind.[294] Das von Ming Li u.a. vorgeschlagene Maß, als „normalized compression distance“ (NCD) bezeichnet, berechnet sich anhand der Längen der komprimierten Zeichenketten wie folgt:[295]

$$sim_{NCD} = \frac{C(XY) - \min(C(X), C(Y))}{\max(C(X), C(Y))}$$

mit

C(XY)	Länge der komprimierten konkatenierten Zeichenketten
C(X)	Länge der komprimierten Zeichenkette X
C(Y)	Länge der komprimierten Zeichenkette Y

[291] Da der Monge-Elkan-Algorithmus jedes Wort der ersten Zeichenkette mit jedem Wort der zweiten Zeichenkette vergleicht, ist er allerdings nicht sonderlich performant

[292] Der Algorithmus von Monge und Elkan sieht eigentlich vor; durch die Anzahl der Wörter der ersten Zeichenkette zu dividieren. Würde die erste Zeichenkette dann Teilmenge der zweiten Zeichenkette sein, so würde der Algorithmus immer eine Übereinstimmung von 100% zurückliefern. In dem in Kapitel 6 vorgestellten Framework wird deshalb durch die Anzahl der Zeichenkette mit den meisten Wörtern dividiert. Alternativ wäre es auch möglich, durch die durchschnittliche Anzahl an Wörtern zu dividieren

[293] Li, M.; Li, X.; Ma, B.; Vitanyi, P.M.B.: The Similarity Metric; erschienen in: Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms; Baltimore; 2003; S. 863-872

[294] Vgl. hierzu: Schulz, R.-H.: Codierungstheorie; 2. Auflage; Vieweg Verlag; Wiesebaden; 2003, S. 39

[295] Cilibrasi, R.; Vitanyi, P.M.B.: Clustering by Compression; erschienen in: IEEE Transactions on Information Theory, Vol. 51, Issue 4; o.O.; 2005; S. 1529

Zur Komprimierung wird ein verlustloses Komprimierungsverfahren verwendet. T. Christen und T. Church ermitteln *C(XY)*, indem X und Y als XY und als YX konkateniert und aus den berechneten Längen der komprimierten Zeichenketten der Mittelwert gebildet wird.[296]

3.2.3 Tokenizer

Bevor Zeichenketten analysiert und verarbeitet werden, kann es sinnvoll sein, diese in linguistische Einheiten wie z.B. Wörter zu segmentieren, um die Ähnlichkeit zwischen Werten besser zu erkennen.[297] Segmentierungsverfahren, die Zeichenketten in Wörter aufteilen, bezeichnet man als Tokenizer.[298] Die einfachste Form eines Tokenizers teilt z.B. eine Zeichenkette an den Stellen auf, wo bestimmte Trennzeichen auftreten, z.B. Leerzeichen, Punkt, Komma o.ä.

Beispiel: 'Hans Muster, 22222 Hamburg' in { 'Hans', 'Muster', '22222', 'Hamburg'}

N-Gram-Tokenizer teilen Zeichenketten nicht in linguistische Einheiten, sondern in gleich große Teilzeichenketten der Länge N auf. Um Buchstaben am Anfang und am Ende des Textes das gleiche Gewicht wie den anderen Buchstaben zuzuordnen, können am Anfang und am Ende des Textes die sogenannten Rand-N-Grams mit Leerzeichen aufgefüllt werden. Zwischenwort-Symbole wie Leerzeichen, Komma usw. können durch das Symbol # ersetzt werden.[299]

Beispiel: 'Muster' als 3-Gram in {'Mus', 'ust', 'ste', 'ter'} und mit Leerzeichen aufgefüllt {'__M', '_Mu', 'Mus', 'ust', 'ste', 'ter', 'er_', 'r__'}

[296] Christen, P.; Churches, T.: Febrl - Freely extensible biomedical record linkage (Manual Release 0.3); http://sourceforge.net/project/downloading.php?groupname=febrl&filename=febrldoc-0.3.pdf; 2005; S. 66
[297] Siehe hierzu auch Abschnitt 3.2.2.3
[298] Vgl. Halama, A.: Flache Satzverarbeitung; erschienen in: Carstensen, K.-U. u.a. (Hrsg.): Computerlinguistik und Sprachtechnologie, 2. Auflage; Spektrum Akademischer Verlag; Heidelberg; 2004; S. 218
[299] Kowalski, G.J.; Maybury, M.T.: Information Storage and Retrieval Systemes - Theory and Implementation, Second Edition; Kluwer Academic; Norwell, Massachusetts; 2000; S. 85

L. Gravano u.a. ergänzen Rand-N-Grams am Anfang und Ende nicht mit Leerzeichen, sondern zur Unterscheidung von Anfang und Ende mit dem #- (Anfang) und mit dem %-Zeichen (Ende).[300]

Beispiel: 'Muster' als 3-Gram mit ‚#' und ‚%' aufgefüllt
{'##M', '#Mu', 'Mus', 'ust', 'ste', 'ter', 'er%', 'r%%'}

Eine Erweiterung von N-Grams sind positionale N-Grams, bei denen zusätzlich die Position des N-Gram in der Zeichenkette angegeben wird,[301] um bei übereinstimmenden N-Grams den Positionsunterschied berücksichtigen zu können.

Beispiel: 'Muster' als positionaler 3-Gram in {{1,'Mus'}, {2,'ust'}, {3,'ste'}, {4,'ter'}}

N-Grams werden u.a. zum Vergleich von Zeichenketten durch tokenbasierte Approximate String Matching Algorithmen eingesetzt[302] indem zwei zu vergleichende Zeichenketten durch einen N-Gram-Tokenizer zerlegt werden. Der Matching Coefficient ergibt sich dann z.B. aus der Anzahl übereinstimmender N-Grams.

Für die beiden Zeichenketten „Muster" und „Naster" ergeben sich folgende 2-Grams:

'_M', 'Mu', 'us', 'st', 'te', 'er', 'r_' und '_N', 'Na', 'as', 'st', 'te', 'er', 'r_'

Beide Zeichenketten haben sieben 2-Grams, wovon 4 sich als Schnittmenge ergeben. Als Jaccard-Maß mit 2-Grams erhält man hier z.B. eine Ähnlichkeit von 40%.

N-Grams haben im Vergleich zu den anderen Approximate String Matching Algorithmen den Vorteil, dass sie vor dem eigentlichen Vergleich erzeugt werden können und damit direkt

[300] Gravano, L.; Ipeirotis, P.G.; Jagadish, H.V.; Koudas, N.; Muthukrishnan, S.; Pietarinen, L.; Srivastava, D.: Using q-grams in a DBMS for Approximate String Processing; erschienen in: IEEE Data Engineering Bulletin, Vol. 24, No. 4; IEEE Computer Society; Washington, DC; 2001; S. 30
[301] Elmagarmid, A.K.; Ipeirotis, P.G.; Verykios, V.S.: Duplicate Record Detection: A Survey; erschienen in: IEEE Transactions on Knowledge and Data Engineering, Vol. 19, No. 1; IEEE Computer Society; Washington, DC; 2007; S. 4
[302] N-Grams werden deshalb normalerweise als Approximate String Matching Verfahren beschrieben, grundsätzlich handelt es sich aber um einen Tokenizer, der zwei Zeichenketten in Mengenelemente aufteilt. Auf diese Elemente kann dann jedes tokenbasierte Verfahren angewendet werden.

indexierbar sind.[303] Dadurch lassen sie sich z.B. auch für das Blocking bei der Duplikaterkennung, zur Rechtschreibkorrektur[304], für die direkte Duplikaterkennung oder für Häufigkeitsverteilungen ähnlicher Wörter in relationalen Datenbanken einsetzen, ohne dass ein direkter Vergleich von zwei Zeichenketten vorgenommen werden muss.

Auch zur Erkennung der Sprache eines Textes können N-Grams geeignet sein, indem die Häufigkeit von N-Grams für einen deutschen Text ermittelt wird und die häufigsten als Indikator für die Sprache oder die Entität gelten.[305] Abhängig von der Erkennung der Sprache könnten dann wieder automatisch sprachspezifische phonetische Codes oder Stemming-Algorithmen verwendet werden. Zu überprüfen bleibt, ob dieser Ansatz auch zur Erkennung der Herkunft von Eigennamen oder Entitäten im Rahmen der „Named Entity Recognition" geeignet ist.

Ein Tokenizer, der systematisch nach bestimmten Zeichen eine Zeichenkette in einzelne Wörter zerlegt, erzeugt normalerweise Wörter, deren Semantiken nicht mehr ohne den Kontext erkennbar sind. Die Zeichenkette „Hr. S. Muster, Bad Homburg, Produktpreis: 100.00 Dollar" kann z.B. in die Wörter „Hr |S |Muster |Bad |Homburg |Produktpreis| 100 |00 |Dollar" zerlegt werden. Hierbei geht die Semantik des Preises, des Ortes und der Person verloren. Aktuellere Ansätze verwenden maschinelle Lernverfahren wie das Maximum Entropy Verfahren, damit ein Tokenizer Wortsegmentierungen besser erkennt.[306]

3.2.4 Nachschlagetabellen

Über Nachschlagetabellen (Lookup-Tables) können Werte domänenspezifisch in einen anderen konvertiert, kategorisiert oder um zusätzliche Werte ergänzt werden. Ein einfaches Beispiel besteht z.B. im Nachschlagen des Ortes, um die Spalte in der entsprechenden Spalte zu ergänzen, zu vereinheitlichen oder zu korrigieren. Aber auch der umgekehrte Weg, das

[303] Heller, M.: Approximative Indexierungstechniken für historische deutsche Textvarianten; erschienen in: Historical Social Research / Historische Sozialforschung, Vol. 31 (2006), No. 3; Zentrum für Historische Sozialforschung e.V.; Köln; 2006; S. 294

[304] Kowalski, G.J.; Maybury, M.T.: Information Storage and Retrieval Systemes - Theory and Implementation, Second Edition; Kluwer Academic; Norwell, Massachusetts; 2000; S. 86

[305] Vgl. hierzu: Beesley, K. R.: Language identifier: A computer program for automatic natural-language identification on on-line text; erschienen in: Proceedings of the 29th Annual Conference of the American Translators Association;1988; S. 47-54
Kowalski, G.J.; Maybury, M.T.: Information Storage and Retrieval Systemes - Theory and Implementation, Second Edition; Kluwer Academic; Norwell, Massachusetts; 2000; S. 86

[306] Vgl. hierzu: http://opennlp.sourceforge.net und http://www.codeplex.com/sharpnlp

Zuweisen eines Wortes zu einer allgemeinen Kategorie (z.B. Kategorie Vorname, Ort) ist sinnvoll, um z.B. Ausreißer in der Repräsentation der Daten zu finden. Wurden in einer Spalte mit Eigennamen diese immer im Format „Hans Muster“ eingegeben, so kann über eine Analyse der zugehörigen Kategorien (Vorname und Nachname) herausgefunden werden, ob Vor- und Nachname z.B. auch in umgekehrter Reihenfolge eingegeben wurden. Beim Parsing[307] kann dieses Verfahren verwendet werden, um einzelne Spalten mit multiplen in Spalten mit atomaren Werten aufzuteilen. Vor allem weitergehende Verfahren der Informationsextraktion (Information Extraction, IE)[308] sind geeignet, solche Kategorien oder Entitäten zu erkennen. Hier sind zum einen Parts-Of-Speech (POS) Tagger zu nennen, die jedes Wort oder Symbol eines Textes mit einem Parts-Of-Speech Tag (Nomen, Verb, Adjektiv usw.) versehen[309] und vor allem dem grammatikalischen Markieren von Worten dienen. Traditionelle POS Tagger unterscheiden acht Kategorien mit weitgehend standardisierten Abkürzungen für die Tags.[310] Der Einsatz von Parts-Of-Speech Tagger ist aber hauptsächlich sinnvoll bei unstrukturierten Textdokumenten.

Sinnvoller einsetzbar im Rahmen der Datenqualitätsverbesserung in relationalen Datenbankmanagementsystemen sind Verfahren der „Named Entity Recognition“ (NER) zur Erkennung von vordefinierten Kategorien. NER hat seinen Ursprung in der „Message Understanding Conference - 6“ (MUC-6) und wurde in der MUC-7 definiert.[311] Diese „Message Understanding Conferences“ hatten in den 90er Jahren das Ziel, Verfahren zur Informationsextraktion aus natürlichsprachlichen Texten zu bewerten.[312] Insgesamt werden bei der NER (auch als Proper Name Classification bezeichnet) acht Kategorien unterschieden: Person, Ort (Location), Organisation, Datum, Zeit, Prozentanteil, Geldwert und sonstige,[313] die innerhalb eines Textes als SGML-Markups gekennzeichnet werden.[314] Zur Erkennung der Entitäten werden verschiedene Verfahren verwendet. Zum einen kommen Nachschlagetabellen in

[307] siehe Abschnitt 3.4.1 Standardisieren
[308] Informationsextraktion gehört zu dem Bereich des Natural Language Processing (NLP) und lokalisiert und identifiziert spezifizierte Teile von Daten eines Dokuments (vgl. hierzu: Califf, M. E.: Relational Learning Techniques for Natural Language Information Extraction, Dissertation;University of Texas; Austin; 1998; S. 5)
[309] Califf, M. E.: Relational Learning Techniques for Natural Language Information Extraction, Dissertation;University of Texas; Austin; 1998; S. 15
[310] Manning, C. D., Schütze, H.: Foundations of Statistical Natural Language Processing; The MIT Press; Cambridge, Massachusetts; 2003; S. 82
[311] Chinchor, N.: MUC-7 Named Entity Task Definition; Version 3.5; http://www.itl.nist.gov/iaui/894.02/related_projects/muc/proceedings/ne_task.html; 1997
[312] Chieu, H.; Ng, H.: Named Entity Recognition with a Maximum Entropy Approach; erschienen in: Proceedings of the Seventh Conference on Natural Language Learning; Edmonton, Canada; 2003; S. 1
[313] Borthwick, A.: A Maximum Entropy Approach to Named Entity Recognition; Ph.D. Thesis; New York University; New York; 1999; S. 1
[314] Grishman, R.; Sundheim, B.: Message Understanding Conference - 6: A Brief History; erschienen in: Proceedings of the 16th conference on Computational linguistics; Copenhagen, Denmark ; 1996

Kombination mit Regeln zum Einsatz.[315] Dabei kann man unterscheiden zwischen internen und externen Beweisen. So haben z.B. die Wörter „Hans Muster“ eine interne Struktur, die sie als Entität Person identifizieren. Es kann sich aber auch um eine Organisation („Hans Muster GmbH“) oder eine Location („Hans Muster Str.“) handeln. Externe kontextabhängige Wörter bestimmen dann letztendlich, welche Entität sich dahinter verbirgt.[316] Zum anderen werden maschinelle Lernverfahren wie Hidden Markov Modelle oder Maximum Entropy Ansätze zur Erkennung der Kategorien verwendet.[317]

Um die Semantik beim Vergleich von zwei Werten zu berücksichtigen, können Ontologien, Thesauri und Domänenwissen verwendet werden, um die Ähnlichkeit zwischen zwei Werten zu erkennen oder zu verfeinern.[318] Unter einer Ontologie versteht man in der Informatik ein Modell einer Anwendungsdomäne, in dem die Objekte der Domäne und deren Wissensbeziehungen in Form eines Netzes von Hierarchien und Beziehungen beschrieben werden. Dabei muss das Modell formal und genau sein und alle Begriffe müssen innerhalb der Benutzergruppe, die das Modell verwenden, „eindeutig und unumstritten definiert sein“.[319] In einer Tabelle ‚Artikel‘ ist z.B. die Farbe des Artikels aufgeführt. Erscheint das gleiche Produkt nun zweimal in der Tabelle (mit der Farbe blau und mit türkis), kann über ein Ontologienetz nun die Ähnlichkeit zwischen den beiden Begriffen blau und türkis ermittelt werden, da es sich bei beiden Farben um Blautöne handelt. Ein Thesaurus kann als eine einfache Form einer Ontologie betrachtet werden.

3.2.5 Reguläre Ausdrücke

Reguläre Ausdrücke benutzen eine Syntax zur Beschreibung von Mustern in Zeichenketten. Die Notation von regulären Ausdrücken unterscheidet zwischen Metazeichen, die einer

[315] Vgl. hierzu: Toral, A.: DRAMNERI: a free knowledge based tool to Named Entity Recognition; erschienen in: Proceedings of the 1st Free Software Technologies Conference; A Coruña, Spanien; 2005; S. 27-32
[316] Mikheev, A.; Moens, M.; Grover, C.: Named Entity Recognition without Gazetteers; erschienen in: Proceedings of the Ninth Conference of the European Chapter of the Association for Computational Linguistics (EACL); Bergen, Norway; 1999; S. 3f
[317] Vgl. hierzu: Borthwick, A.: A Maximum Entropy Approach to Named Entity Recognition; Ph.D. Thesis; New York University; New York; 1999 und
Klein, D.; Smarr, J.; Nguyen, H.; Manning, C.: Named Entity Recognition with Character-Level Models; erschienen in: Proceedings of the Seventh Conference on Natural Language Learning; Edmonton, Canada; 2003
[318] Müller, H.; Weis, M.; Bleiholder, J.; Leser, U.: Erkennen und Bereinigen von Datenfehlern in naturwissenschaftlichen Daten; erschienen in: Datenbank-Spektrum, 5. Jahrgang, Heft 15; dpunkt Verlag, Heidelberg; 2005; S. 29
[319] Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 269

bestimmten Semantik entsprechen, und Literalen. Die am häufigsten verwendeten Metazeichen können der Tabelle 13 entnommen werden.

Tab. 13: Metazeichen bei regulären Ausdrücken

Metazeichen	**Semantik**	**Match**
.	Punkt	Beliebiges Zeichen
[…]	Zeichenklasse	Eins der zwischen den eckigen Klammern aufgelisteten Zeichen
[^…]	Negierte Zeichenklasse	Eins der nicht zwischen den eckigen Klammern aufgelisteten Zeichen
Zeichen	Escape	*Zeichen* ist normalerweise ein Metazeichen
?	Fragezeichen	Darf einmal auftreten, optional
*	Stern	Keinmal oder beliebig
+	Plus	Einmal oder beliebig
{min, max}	Expliziter Bereich	Bereichsbegrenzung des Auftretens
^	Zirkumflex	Zeilenanfang
$	Dollar	Zeilenende
\<	Wortgrenze	Wortanfang
\>	Wortgrenze	Wortende
|	Alternation	eine der aufgeführten Alternativen
(…)	Klammern	beschränkt Geltungsbereich von Alternationen; gruppiert für nachfolgende Quantoren; Text für Rückwärtsreferenzen merken
\1, \2, …	Rückwärtsreferenzen	Text, der im ersten, zweiten usw. Paar von Klammern auftritt

Quelle: eigene Darstellung in Anlehnung an: Friedl, J.E.F.: Reguläre Ausdrücke, 2. Auflage; O'Reilly; 2003; Köln; S. 33

Reguläre Ausdrücke eignen sich z.B. zur Überprüfung der Repräsentation von Daten. So besteht ein einfacher regulärer Ausdruck zur Überprüfung von Postleitzahlen z.B. aus der Zeichenkette `"^(D-)?[0-9]{5}".`

3.2.6 Segmentierung

Beim Clustering bzw. bei der Segmentierung geht es um die Einteilung von Objekten in natürliche Gruppen,[320] die zum einen weitgehend homogen, zum anderen aber auch weitgehend trennbar sein sollen.[321] Im Bereich der Datenqualität kann die Segmentierung vor allem zur Analyse von Ausreißern verwendet werden.[322]

[320] Witten, I.H.; Frank, E.: Data Mining; Hanser Verlag; München, Wien; 2001; S. 229
[321] Bortz, J.: Statistik für Human- und Sozialwissenschafter; 6. Auflage; Springer Medizin Verlag; Heidelberg; 2005; S. 565
[322] Vgl. hierzu Abschnitt 3.4.1

3.2.7 Klassifizierer

Klassifizierer sind Algorithmen, die den Wert eines bestimmten abhängigen Attributes auf Basis einer oder mehrerer unabhängiger Attribute vorhersagen. Algorithmen von Klassifizierern gibt es eine Vielzahl: Neben statistischen Verfahren wie der Regressionsanalyse wurden vor allem im Bereich der Informatik maschinelle Lernverfahren, wie Neuronale Netze oder Entscheidungsbäume entwickelt. Neben dem Algorithmus unterscheiden die Verfahren sich zum einen in der Transparenz. So ist der Lösungsweg bei Entscheidungsbäumen im Gegensatz zu neuronalen Netzen unmittelbar nachvollziehbar. Zum anderen existieren Unterschiede bei abhängigen und unabhängigen Attributen. Neuronale Netze benötigen ausschließlich numerische Attribute als unabhängige Variable, ein einfacher Entscheidungsbaum wie 1R verwendet dagegen sowohl nominale als auch numerische Attribute. Numerische Attribute können jedoch durch Diskretisierung in nominale Attribute umgewandelt werden.

Im Folgenden soll am Beispiel des einfachen 1R-Algorithmus die Funktion eines Klassifizierers beschrieben werden. Bei den Beispieldaten soll die Bonität von den unabhängigen Attributen Altersgruppe und Kinder vorhergesagt werden.

Tab. 14: Beispieldaten zur Klassifizierung

Kundennr.	Altersgruppe	Kinder	Bonität
1	junior	Nein	Nein
2	junior	Nein	Nein
3	junior	Nein	Nein
4	senior	Ja	Ja
5	junior	Nein	Ja
6	senior	Nein	Ja

Quelle: eigene Darstellung

Für jedes Attribut wird überprüft, inwieweit es alleine zur Vorhersage geeignet ist. Dabei wird ermittelt, wie hoch der Fehleranteil an Falschvorhersagen ist. Dazu werden die Häufig-

keiten der Ausprägungen des jeweils ausgewählten unabhängigen und des abhängigen Attributs errechnet. Die Ausprägung bzw. der Wert des abhängigen Attributs, der am häufigsten auftritt, wird dann zur Vorhersage verwendet. So wird z.B. für das Attribut Altersgruppe die Bonität „Nein“ am häufigsten korrekt vorhergesagt, wenn der Kunde zur Altersgruppe „junior“ gehört (3mal).

Tab. 15: Vorgehen beim 1R-Algorithmus

Vorhersage bei Attribut Altersgruppe

Wertekombination		Häufigkeit	Fehleranteil
Altersgruppe	Bonität		
junior	Nein	3	1/4
junior	Ja	1	
senior	Nein	0	
senior	Ja	2	0/2
Fehlerquote bei Vorhersage			1/6

Vorhersage bei Attribut Kinder

Wertekombination		Häufigkeit	Fehleranteil
Kinder	Bonität		
Nein	Nein	3	2/5
Nein	Ja	2	
Ja	Nein	0	
Ja	Ja	1	0/1
Fehlerquote bei Vorhersage			2/6

Quelle: eigene Darstellung

Für jedes Attribut werden die Einzelfehlerquoten zu einer Fehlerquote summiert und das Attribut mit der geringsten Quote zur Vorhersage verwendet. Der 1R-Algorithmus würde für das obige Beispiel das Attribut Altersgruppe wählen und zwei Klassifizierungsregeln bilden:

IF Altersgruppe = junior THEN Bonität = Nein
IF Altersgruppe = senior THEN Bonität = Ja

Auf Grundlage der obigen Daten liegt die Wahrscheinlichkeit, dass die Bonität korrekt vorhergesagt wird, bei etwa 83% (5/6).

3.2.8 Regeln

3.2.8.1 Einführung

Bis heute werden in der Anwendungsentwicklung Geschäftsregeln weitgehend in Softwarekomponenten „verborgen“ implementiert, obwohl relationale Datenbankmanagementsysteme (RDBMS) durchaus eine Vielzahl an Integritätsbedingungen bieten, um Geschäftsregeln abzubilden. Bei relationalen Datenbankmanagementsystemen liegt ein Grund hierfür sicherlich darin, dass diese eher eine passive und aktive Eigenschaften (abgesehen von Triggern) eher eine nachrangige Rolle spielen.[323] Das führt dazu, dass Änderungen an den Regeln nur aufwendig durchzuführen sind.

Eine Entwicklungstendenz zur Behebung dieses Problems sind Business Rules Engines (BRE). Sie organisieren und verwalten Geschäftsregeln in einem einheitlichen Repository und bieten eine Wissensbasis für Mitarbeiter in der Organisation. Das Konzept der Geschäftsregeln soll die Möglichkeit bieten, die Logik eines Unternehmens zentral zu verwalten und für jeden Mitarbeiter verständlich und erkennbar abzubilden. Zudem können Anwendungen, die auf der Business Rules Engine aufsetzen, flexibel und schnell geändert werden und sind durch die Geschäftsregeln in ihrer Logik einfacher überprüfbar.[324] Ein weiterer wichtiger Ansatzpunkt bei BREs ist die Ausrichtung auf den Anwender. Geschäftsregeln sollten im BRE nicht technikorientiert verwaltet, sondern vielmehr von den Mitarbeitern aus den Fachbereichen in Zusammenarbeit mit der IT-Abteilung in verständlicher Form definiert und transparent gemacht werden.

Regeln bzw. im engeren Sinne Geschäftsregeln treten in verschiedenen Variationen auf: Sie können Einschränkungen festlegen (Constraint Rules), Vorschläge machen (Guideline Rules), Berechnungen durchführen (Computation Rules) und Aktionen durchführen (Action-Enabler Rules).[325] Eine Business Rule Engine sollte das Finden und strukturierte Ablegen von Regeln

[323] Chisholm, M.: How to build a Business Rules Engine; Morgan Kaufmann; San Francisco; 2004; S. xvii
[324] Chisholm, M.: How to build a Business Rules Engine; Morgan Kaufmann; San Francisco; 2004; S. xxf
[325] von Halle, B.; Goldberg, L.: The Business Rule Revolution; Happy About; Silicon Valley; 2006; S. 8

unterstützen, da deren Anzahl normalerweise sehr hoch ist.[326] Außerdem sollten Regeln zentral und für jeden in der Organisation zugreifbar gespeichert werden (Transparenz). Die Regeln sollten durch das BRE nachverfolgbar sein, d.h. in welchem Anwendungssystem, in welchem Prozess, bei welcher Entscheidung und in welchem Anwendungsfall bezogen auf welches Datenattribut sie angewendet werden.[327] Regeln sollten aber auch nicht zu restriktiv sein. Das könnte dazu führen, dass bei der Eingabe z.B. Daten nicht oder aber falsch eingegeben werden, nur um regelkonform zu sein.[328] Zu Regeln gehört nicht nur das Spezifizieren und Ausführen dieser, sondern auch, dass die Regelbasis nach Änderung oder Einfügen neuer Regeln auf Inkonsistenzen und Redundanzen untersucht wird und dass eine Regelverarbeitung immer terminiert und zu einem konsistenten Datenbankzustand führt.[329]

3.2.8.2 Regelspezifikation

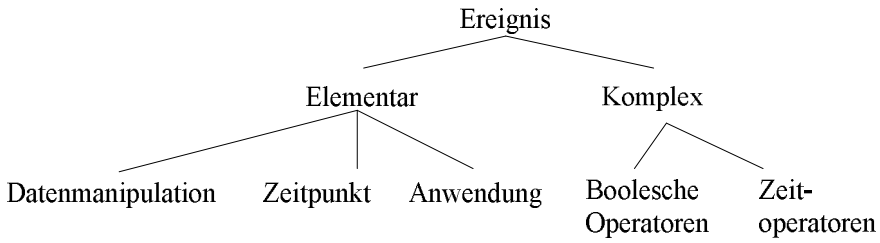
Regeln werden in dieser Arbeit in Anlehnung an aktive Datenbankmanagementsysteme in drei Bereiche gegliedert:

- Ereignis
- Bedingung
- Aktion

Verändert sich ein bestimmter Status, so tritt ein Ereignis ein und eine bestimmte Bedingung kann überprüft werden. Tritt diese Bedingung ein, sind Voraussetzungen erfüllt, so dass eine bestimmte Aktion ausgeführt werden kann bzw. muss. Ereignisse lassen sich unterteilen in elementare und komplexe Ereignisse.

[326] von Halle, B.; Goldberg, L.: The Business Rule Revolution; Happy About; Silicon Valley; 2006; S. 166
[327] Vgl. hierzu: von Halle, B.; Goldberg, L.: The Business Rule Revolution; Happy About; Silicon Valley; 2006; S. 8
[328] Lee, Y.W.; Pipino, L.L.; Funk, J.D.; Wang, R.Y.: Journey to Data Quality; MIT Press; Cambridge, Massachusetts; 2006; S. 81
[329] Schlesinger, M.: ALFRED - Konzepte und Prototyp einer aktiven Schicht zur Automatisierung von Geschäftsregeln, Dissertation; Rechts- und Wirtschaftswissenschaftliche Fakultät der Universität Bern; Bern; 2000; S. 113

Abb. 24: Ereignisse



Quelle: eigene Darstellung in Anlehnung an: Herbst, H.; Knolmayer, G.: Ansätze zur Klassifikation von Geschäftsregeln, Arbeitsbericht Nr. 46; Institut für Wirtschaftsinformatik der Universität Bern; Bern; 1994; S. 6

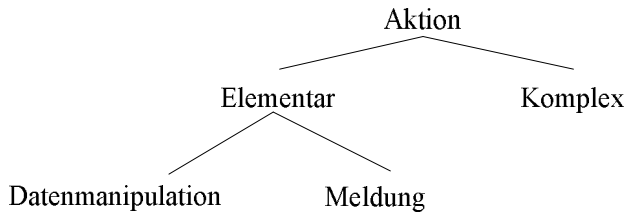
Komplexe Ereignisse ergeben sich aus elementaren Ereignissen, die über boolesche Operatoren oder Zeitoperatoren zusammengesetzt werden. Zu den elementaren Ereignissen gehören Datenmanipulationsereignisse, die auftreten, wenn Daten sich ändern oder aber abgefragt werden (um z.B. die Qualität von Abfragen zu analysieren). Zeitpunktereignisse treten zu einem bestimmten Datum und einer bestimmten Uhrzeit ein. Anwendungsereignisse dagegen werden nicht automatisch, sondern von einer Anwendung manuell durch Programmcode ausgelöst. Neben den komplexen Ereignissen mit booleschen Operatoren kann man Zeitoperatoren unterscheiden. Hierbei kann es sich um Intervall-, Perioden- oder Verzögerungsereignisse handeln. Bei einem Intervallereignis tritt ein spezifiziertes Ereignis innerhalb eines Zeitraumes ein (z.B. Kunde nimmt Angebot innerhalb von 30 Tagen an). Ein Periodenereignis wird immer wiederkehrend nach einer bestimmten Zeitspanne ausgeführt (z.B. jeden 1.ten des Monats Mahnungen versenden). Ein Verzögerungsereignis schließlich tritt auf, wenn nach einem Ereignis eine spezifizierte Zeit abgelaufen ist (z.B. 30 Tage nach Fakturierung Rechnungseingang überprüfen).

Bei einer Regel kann man wie bei den Ereignissen auch zwischen elementaren und komplexen Bedingungen unterscheiden. Komplexe Bedingungen setzen sich dabei durch Verknüpfung elementarer Bedingungen durch boolesche Operatoren zusammen.

Aktionen lassen sich in elementare und komplexe Aktionen unterteilen. Bei den elementaren Aktionen kann man Datenmanipulations- und Meldungsaktionen unterscheiden. Datenmanipulationsaktionen ändern Daten auf Grundlage des eingetretenen Ereignisses. Meldungs-

aktionen liefern eine Benachrichtigung oder protokollieren das Ereignis, um daraufhin manuelle Aktionen zu initiieren.[330]

Abb. 25: Aktionen



Quelle: eigene Darstellung in Anlehnung an: Herbst, H.; Knolmayer, G.: Ansätze zur Klassifikation von Geschäftsregeln, Arbeitsbericht Nr. 46; Institut für Wirtschaftsinformatik der Universität Bern; Bern; 1994; S. 9

Ereignis, Bedingung und Aktion sind optional, wobei ein Ereignis zumindest immer mit einer Aktion, normalerweise auch mit einer Bedingung zusammen auftreten sollte:

ON UPDATE Geburtsdatum
DO Alter = Tagesdatum – Geburtsdatum

oder

ON UPDATE Geburtsdatum
IF YEAR(Geburtsdatum) > YEAR(Tagesdatum)
DO Meldung

Wird nur die Bedingung angegeben, so handelt es sich um eine Integritätsbedingung, die eine Einschränkung beschreibt. Bezogen auf das letzte Beispiel könnte eine Integritätsbedingung im Pseudocode wie folgt deklariert werden:

YEAR(Geburtsdatum) > YEAR(Tagesdatum)

oder

ALTER < 120

Regeln sollten aktivierbar und deaktivierbar sein. Wichtig in diesem Zusammenhang ist auch die Möglichkeit der Simulation von Regeln vor deren Aktivierung.

[330] Vgl. hierzu: Herbst, H.; Knolmayer, G.: Ansätze zur Klassifikation von Geschäftsregeln, Arbeitsbericht Nr. 46; Institut für Wirtschaftsinformatik der Universität Bern; Bern; 1994; S. 6ff

3.2.8.3 Regelinduktion

Verfahren zur Regelinduktion erkennen automatisch Regeln aus einem Datenbestand, indem auf Grundlage einzelner Datenwerte induktiv allgemeine Regeln ermittelt werden. Die Regelinduktion dient damit als Hilfsmittel zur Definition von Geschäftsregeln. Anhand der folgenden einfachen Daten soll die Regelinduktion erläutert werden:

Tab. 16: Beispieldaten zur Regelinduktion

Kundennr.	Altersgruppe	Bonität
1	jung	mangelhaft
2	jung	mangelhaft
3	senior	sehr gut
4	jung	sehr gut
5	mittleres Alter	sehr gut
6	mittleres Alter	sehr gut

Quelle: eigene Darstellung

Anhand der Datenwerte kann man erkennen, dass in den Fällen, in denen die Altersgruppe gleich „senior“ und „mittleres Alters“, die Bonität „sehr gut“ ist. In der Altersgruppe „jung“ ist die Bonität dagegen bis auf einen Datensatz „mangelhaft“. Durch Regelinduktion könnten nun folgende drei Regeln aufgestellt werden, indem man die zusammen auftretenden Häufigkeiten ermittelt:

IF Alter = „jung“ THEN Bonität = „mangelhaft“
IF Alter = „mittleres Alter“ THEN Bonität = „sehr gut“
IF Alter = „senior“ THEN Bonität = „sehr gut“

Die zweite Regel „mittleres Alter“ trifft für alle Datensätze zu, für die die Bedingung zutrifft. Die dritte Regel „senior“ trifft auch für alle Datensätze mit dieser Bedingung zu, allerdings gibt es hierzu nur einen einzigen Datensatz. Die Aussagekraft der zweiten Regel sollte also höher bewertet werden als die der dritten. Die Bedingung der ersten Regel trifft auf drei

Datensätze zu, wobei das Ergebnis bei dem Datensatz mit der Kundennr. 4 nicht „mangelhaft“, sondern „sehr gut“ ist.

Um nun die Qualität einer Regel beurteilen zu können, benötigt man Kennzahlen, die die Häufigkeit und Signifikanz der Regel kennzeichnen. Die bekanntesten Kennzahlen zur Beurteilung von Regeln sind:

- Support
- Confidence
- Expected Confidence
- Lift

Support beschreibt den Anteil an allen Datensätzen, in denen sowohl die Bedingung (IF-Teil) als auch das Ergebnis (THEN-Teil) zusammen auftreten. Durch Confidence wird der Zusammenhang zwischen der Bedingung (IF-Teil) und dem Ergebnis (THEN-Teil) ausgedrückt.[331] Es wird die Anzahl der Datensätze, in denen die Bedingung und das Ergebnis zutreffen ins Verhältnis gesetzt zur Anzahl der Sätze, in denen nur die Bedingung auftritt. Expected Confidence ist die Anzahl der Datensätze, in denen das Ergebnis auftritt im Verhältnis zur Anzahl aller Datensätze.[332] Lift schließlich stellt das Verhältnis der Kennzahl Confidence zur Kennzahl Expected Confidence dar und misst damit die Korrelation zwischen Bedingung und Ergebnis der Regel. Ein Lift kleiner 1 deutet auf eine negative, ein Lift größer 1 auf eine positive und ein Lift von 1 auf keine Korrelation hin.[333]

Für die aufgestellten Regeln ergeben sich folgende Häufigkeiten und daraus die entsprechenden Kennzahlen:

[331] Hettich, S.; Hippner, H.: Assoziationsanalyse; erschienen in: Hippner, H. u.a. (Hrsg.): Handbuch Data Mining im Marketing, 1. Auflage,;Vieweg; Braunschweig; 2001; S. 429
[332] „Expected Confidence“ entspricht damit dem Support nur für den Ergebnisteil (THEN-Teil)
[333] Hettich, S.; Hippner, H.: Assoziationsanalyse; erschienen in: Hippner, H. u.a. (Hrsg.): Handbuch Data Mining im Marketing, 1. Auflage,;Vieweg; Braunschweig; 2001; S. 447

Tab. 17: Kennzahlen einer Regel

	Bedingung / Ergebnis	Anzahl	Gesamt
Regel 1	jung	3	2
	mangelhaft	2	
Regel 2	mittleres Alter	2	2
	sehr gut	4	
Regel 3	Senior	1	1
	sehr gut	4	
Datensätze gesamt			6

Kennzahl \ **Regel**	**Regel 1**		**Regel 2**		**Regel 3**	
Support	2/6	33%	2/6	33%	1/6	17%
Confidence	2/3	67%	2/2	100%	1/1	100%
Expected Confidence	2/6	33%	4/6	67%	4/6	67%
Lift		2		1,5		1,5

Quelle: eigene Darstellung

Für die Regel 1 lassen sich die Kennzahlen wie folgt interpretieren: 67% der Datensätze mit der Bedingung Altersgruppe „jung“ (3 Datensätze), weisen die Bonität „mangelhaft“ auf. Confidence stellt damit eine bedingte Wahrscheinlichkeit des Ergebnis-Teils dar. Analog interpretiert gilt dann genauso: Ist die Altergruppe „jung“, so beträgt die Wahrscheinlichkeit, dass die Bonität „mangelhaft“ ist, 67%. In 33% aller Datensätze enthält die Altersgruppe den Wert „jung“ und die Bonität den Wert „mangelhaft“. Damit gibt Support den prozentualen Anteil bezogen auf alle Datensätze an, bei denen die Regel korrekt ist.[334] Die Regel 3 trifft ausschließlich auf den Datensatz mit der Kundennummer 3 zu (daher auch der geringe Wert der Kennzahl Support).

Das allgemeine Problem bei der Regelinduktion liegt darin, dass es sehr viele mögliche Regeln geben kann. Ein Lernverfahren zur Regelinduktion sollte deshalb benutzerdefinierte

[334] Vgl. hierzu: Borgelt, C.; Kruse, R.: Induction of Association Rules: Apriori Implementation; erschienen in: 15th Conference on Computational Statistics (Compstat 2002); Berlin; S. 395

Minimumwerte für die Kennzahlen Support und Confidence berücksichtigen,[335] um nicht Regeln zu induzieren, die nur auf wenig Datensätze zutreffen.

Die ersten Algorithmen zur Suche von Assoziationsregeln[336] sind die 1993 und 1994 am IBM Forschungszentrum in Almaden entwickelten Apriori, AprioriTid und AprioriHybrid Verfahren.[337] Diese schränken die Anzahl der Regeln ein, indem sie die Häufigkeiten von Einzelwerten und anschließend von Wertekombinationen ermitteln und diejenigen herausfiltern, die den benutzerdefinierten Support unterschreiten. Daneben existieren neuere Algorithmen wie FP-Growth (Frequent Patterns), Eclat und Recursive Elimination, die effizienter ausgeführt werden.[338]

Die Erzeugung von Assoziationsregeln basiert auf Häufigkeiten der Werte oder Wertekombinationen. Insofern können Assoziationsregeln nur mit nominalen Attributen sinnvolle Ergebnisse liefern. R. Srikant und R. Agrawal erweiterten deshalb 1996 den Apriori-Algorithmus um quantitative Assoziationsregeln, indem numerische Attribute diskretisiert werden.[339] Neben diesem speziell für Assoziationsregeln beschriebenen Verfahren zur Diskretisierung existieren allgemeine Algorithmen zur Diskretisierung numerischer Daten wie z.B. die Verfahren des Binning. Beim Binning werden die Daten sortiert und die sortierten Werte in „Behälter“ (Bins) gleicher Größe verteilt.[340] Ist die Häufigkeit pro Bin gleich, so handelt es sich um Equidepth Binning. Sind die Wertintervalle der einzelnen Bins identisch, spricht man von Equiwidth Binning.[341] Binning verwendet dabei keine Klasseninformationen von einem Klassifizierer und gehört daher zu den unsupervised Diskretisierungsmethoden. Eine einfache supervised Disketrisierungsmethode beschreibt Holte, der die numerischen Werte

[335] Srikant, R.; Agrawal, R.: Mining Quantitative Association Rules in Large Relational Tables; erschienen in: Proceedings of the ACM SIGMOD Conference on Management of Data; Montreal; 1996; S. 1
[336] Ursprünglich wurden diese Algorithmen zur Warenkorbanalyse entwickelt um Assoziationen zwischen gleichzeitig gekauften Produkten aufzudecken
[337] Hettich, S.; Hippner, H.: Assoziationsanalyse; erschienen in: Hippner, H. u.a. (Hrsg.): Handbuch Data Mining im Marketing, 1. Auflage,;Vieweg; Braunschweig; 2001; S. 429
[338] Vgl. hierzu: Borgelt, C.: Efficient Implementations of Apriori and Eclat; erschienen in: Workshop of Frequent Item Set Mining Implementations (FIMI 2003); Melbourne, Florida; 2003
Borgelt, C.: An Implementation of the FP-growth Algorithm; erschienen in: Workshop Open Source Data Mining Software (OSDM'05); Chicago; 2005; S. 1-5
Borgelt, C.: Keeping Things Simple: Finding Frequent Item Sets by Recursive Elimination;erschienen in: Workshop Open Source Data Mining Software (OSDM'05); Chicago; 2005; S. 66-70
[339] Srikant, R.; Agrawal, R.: Mining Quantitative Association Rules in Large Relational Tables; erschienen in: Proceedings of the ACM SIGMOD Conference on Management of Data; Montreal; 1996; S. 1
[340] Han, J.; Kamber, M.: Data Mining - Concepts and Techniques; Morgan Kaufmann; San Francisco; 2001; S. 110
[341] Han, J.; Kamber, M.: Data Mining - Concepts and Techniques; Morgan Kaufmann; San Francisco; 2001; S. 255

abhängig von einer vorherzusagenden Klasse durch einen 1R-Klassifizierer in einzelne Bins verteilt.[342]

Eine Generierung von Regeln zur Erkennung von Problemen mit numerischen Attributen ist sinnvoll, um z.B. Unterschiede in den Differenzen von Attributen zu erkennen (z.B. Bestelldatum muss kleiner Lieferdatum sein, bestellte Menge muss kleiner der vorhandenen Lagermenge sein o.ä.). In Kapitel 6 wird ein einfacher Algorithmus vorgestellt, der solche Regeln generiert.

Neben Regeln mit numerischen Attributen sollten Regeln ermittelt werden können, die sich auf prozentuale Anteile im Datenbestand beziehen (z.B. das Geschlecht weiblich tritt in der Tabelle Kunde zu etwa 45% auf).[343] Die Validierung solcher Regeln setzt natürlich ausreichende Kenntnisse über die Gesamtheit der Daten voraus.

3.3 Stichprobenentnahme (Sampling)

Bestimmte Verfahren zur Datenqualitätsverbesserung wie z.B. die Regelinduktion oder Klassifizierung, sind sehr rechen- und ressourcenaufwendig. Bei großen Datenbeständen ist es deshalb sinnvoll, Stichproben zu entnehmen und die Verfahren innerhalb dieser Stichproben durchzuführen.

In der Statistik wird zwischen Wahrscheinlichkeitsstichprobe und Nicht-Wahrscheinlichkeitsstichprobe unterschieden. Die Wahrscheinlichkeitsstichprobe wird im Gegensatz zur Nicht-Wahrscheinlichkeitsstichprobe auf Basis einer bekannten Wahrscheinlichkeit gezogen. Deshalb kann bei der Nicht-Wahrscheinlichkeitsstichprobe die Reliabilität nicht berechnet werden.

Beim zufallsbasierten Verfahren werden aus einer Tabelle mit N Datensätzen n Datensätze als Stichprobe gezogen, indem eine Serie von n Zufallszahlen von 1 bis N erzeugt wird. Beim systematischen Stichprobenverfahren, wird zunächst zufallsbasiert ein Datensatz als Start-

[342] Dougherty, J.; Kohavi, R.; Sahami, M.: Supervised and Unsupervised Discretization of Continuous Features; erschienen in: Machine Learning: Proceedings of the 12th International Conference; Morgan Kaufmann; San Francisco; 1995; S. 197-202

[343] Helfert, M.: Proaktives Datenqualitätsmanagement in Data-Warehouse-Systemen, Dissertation; logos-Verlag; Berlin; 2002; S. 170

punkt ermittelt und daraufhin jeder k-te Datensatz für die Stichprobe verwendet. Ist die Verteilung der Datensätze nicht gleichförmig, werden andere Verfahren verwendet, die die Datenbasis z.B. zunächst in mehrere Datengruppen aufteilt.[344]

3.4 Verstehen

3.4.1 Grundlagen

Nach der Identifikation aller kritischen Geschäftsprozesse und deren Informationsketten, werden diese mit den dazugehörigen Daten analysiert, um Probleme im Ablauf und in der Datenspeicherung zu erkennen. Es folgt eine Festlegung der zu messenden Datenqualitätsmerkmale, sowie die Art und Durchführung der Messung. Zum Identifizieren gehören auch erste Überlegungen, wie problematische Geschäftsprozesse und deren Informationsketten verbessert werden können, um Probleme zu beseitigen.

Ziel des Verstehens der Daten ist das Erkennen der eigentlichen Semantik dieser im Vergleich zu der einmal im Entwurf festgelegten. Semantische Unterschiede hierbei sind Indizien für Datenqualitätsprobleme bzw. für eine andere Verwendung von Daten als im Entwurf vorgesehen. Eine Analyse und „wahre“ Beschreibung im Sinne der vorhandenen Daten führt zu einer Einschätzung der Datenqualität und zu ersten Maßnahmen zur Verbesserung.

3.4.2 Analysieren

Das Analysieren besteht aus drei wesentlichen Teilaufgaben:

1) Analyse der Datenstruktur
2) Analyse der Dateninhalte
3) Analyse der Beziehungen

Vor einer Qualitätsverbesserung, ist es entscheidend, die Zusammenhänge zwischen Geschäftsprozessen, Daten, Datenfluss und Datenqualitätsmerkmalen verstanden zu haben.[345]

[344] Lee, Y.W.; Pipino, L.L.; Funk, J.D.; Wang, R.Y.: Journey to Data Quality; MIT Press; Cambridge, Massachusetts; 2006; S. 70f

Beim Analysieren versucht ein Domänenexperte daher, mögliche Probleme in den Daten, der Datenstruktur und in den Datenbeziehungen zu erkennen. Das Analysieren der Daten kann auf zwei Arten erfolgen. Entweder hat der Domänenexperte eine Vermutung eines Datenqualitätsproblems und versucht dieses gezielt zu verifizieren, oder er versucht neue Probleme zu entdecken.

Die Datenstruktur wird in der Regel aus den Metadaten der relationalen Datenbank ermittelt. Die Datenwerte selbst liefern aber auch Informationen zur Datenstruktur. So kann z.B. die Eigenschaft ‚Datentyp' aus den Metadaten der relationalen Daten ermittelt werden oder aber durch Analyse der Dateninhalte. Bei der Eigenschaft Länge eines Attribut kann z.B. analysiert werden, ob diese zu klein definiert wurde, indem überprüft wird, wie hoch der Anteil der Zeichenketten ist, die die Länge vollständig ausnutzen. Die wichtigsten Strukturinformationen sind Datentyp, Länge, Dezimalstellen und Wertebereichsdefinition.

Neben der Anzeige und manuellen Durchsicht der Daten sollte auf eindeutige Attributbezeichnung, Wertebereichsdefinition, Datentyp, Kardinalität, Häufigkeiten, Muster (Pattern), fehlende Werte, Eindeutigkeit der Werte, Geschäftsregeln sowie statistische Werte wie Maximum, Minimum, Modalwert, Median usw. geprüft werden. Über statistische Methoden wie die Berechnung arithmetischer Mittelwerte, Standardabweichung, Varianz usw. kann ein Domänenexperte unerwartete Indikatoren erkennen und damit mögliche ungültige Datensätze erkennen. Auch Kennzahlen zur Verteilung der Daten können Auskunft über Mängel geben (Gauß'sche Normalverteilung, Schiefe, Wölbung). Bei numerischen Attributen werden vor allem statistische Kennzahlen, wie Minimum, Maximum, Mittelwert, Standardabweichung usw. betrachtet. Um bei der Berechnung von statistischen Kennzahlen Ausreißer weitgehend zu ignorieren, kann ein prozentualer Anteil der Werte im unteren und oberen Bereich abgeschnitten werden. T. Dasu und T. Johnson bezeichnen einen Mittelwert, der so berechnet wird, dann als getrimmten Mittelwert („Trimmed Mean").[346]

Die Kardinalität kann zur Einschätzung des Datenqualitätsmerkmals Eindeutigkeit verwendet werden. So muss bei einem Primärschlüssel die Kardinalität immer der Anzahl der Datensätze entsprechen. Außerdem kann sie zu einer ersten Überprüfung der Wertebereiche verwendet

[345] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 219
[346] Dasu, T.; Johnson, T.: Exploratory Data Mining and Data Cleaning; John Wiley & Sons; Hoboken, New Jersey; 2003; S. 29

werden. Die Kardinalität einer Spalte zur Speicherung des Bundeslandes sollte z.B. nicht größer als 16 sein.

Soll z.B. die Telefonnummer immer im Format „99999 / 999999-999“ eingegeben werden, so kann über eine Muster- oder Patternanalyse herausgefunden werden, ob dieses Muster bei jeder Telefonnummer angewendet wurde.

Auch Häufigkeitsverteilungen von Codes[347] können bei der Analyse Datenqualitätsprobleme aufdecken. So können z.B. über die Anwendung eines phonetischen Codes häufig falsch erfasste Wert ermittelt werden.

Ausreißer sind in der Statistik stark von anderen abweichende Werte.[348] Sie stellen somit eventuell falsche Datenwerte dar und können u.a. statistische Kennzahlen ungünstig verändern. Hierzu ein Beispiel: Das Attribut Name enthält zu 20% den Wert „Hans Muster“ oder eine Reihe von Zahlen enthält zu 95% Zahlen unter 100 und zu 5% Zahlen über 1.000.

Zur Erkennung von Ausreißern können neben Cluster- bzw. Segmentierungsverfahren auch Häufigkeitsverteilungen von Werten (maximaler/minimale Werte) oder von Mustern dienen. Ausreißer können einen inkorrekten Wert darstellen oder repräsentieren einen selten vorkommenden Wert, der aber korrekt ist. Sind Ausreißer erkannt, muss in einem nächsten Schritt überprüft werden, ob es sich um einen inkorrekten Wert handelt. Entsprechend muss es bei der Analyse immer die Möglichkeit eines Drill-Downs der Daten geben. Hat man bei einer Häufigkeitsverteilung von Mustern Ausreißer entdeckt, so sollte eine Überprüfung der dahinterliegenden Daten erfolgen, um tatsächliche Datenqualitätsmangel aufzudecken. Ausreißer werden ausführlich von T. Dasu und T. Johnson behandelt.[349]

Bei der Segmentierung bzw. beim Clustern wird für jedes Datensatzpaar ein aggregiertes Distanz- oder Ähnlichkeitsmaß (Proximitätsmaß) errechnet, um ähnliche Datensätze zu Datengruppen, sogenannten Clustern oder Segmenten, zusammenzufassen. Cluster mit nur wenigen Datensätzen weisen dann auf eventuelle Ausreißer hin. Außerdem können Daten-

[347] Vgl. Abschnitt 3.2.1
[348] Kübel, J.; Grimmer, U.; Hipp, J.: Regelbasierte Ausreißersuche zur Datenqualitätsanalyse; erschienen in: Datenbank-Spektrum, 5. Jahrgang, Heft 14; dpunkt Verlag; Heidelberg; 2005; S. 22
[349] Dasu, T.; Johnson, T.: Exploratory Data Mining and Data Cleaning; John Wiley & Sons; Hoboken, New Jersey; 2003; S. 146ff

sätze, die weit vom Zentrum ihres jeweiligen Clusters entfernt sind, als Ausreißer identifiziert werden.

Soll überprüft werden, ob ein Attribut in einer Tabelle überhaupt verwendet wird, sollte die Anzahl fehlender Werte (bei relationalen Datenbanken NULL) in Bezug zur Anzahl aller Datensätze überprüft werden.[350] Die Eindeutigkeit (Uniqueness) kann als Verhältnis zwischen der Anzahl unterschiedlicher Werte in Relation zur Anzahl aller Datensätze berechnet werden. Ist also die Kardinalität gleich der Anzahl Datensätze, so ist die Eigenschaft Eindeutigkeit zu 100% gegeben.

Auch die Verteilung des Auftretens der ersten Ziffer nach F. Benford kann Rückschlüsse über die Korrektheit der Daten geben. Das Benford'sche Gesetz beschreibt die Tatsache, dass die Zahlen von 1 bis 9 als erste Ziffer von numerischen Werten unterschiedlich oft auftreten, also nicht gleich verteilt sind.[351] Diese schon 1881 von S. Newcombe und später unabhängig davon von F. Benford beobachtete Eigenschaft wurde empirisch von F. Benford in der folgenden Verteilungstabelle dargelegt:[352]

Tab. 18: Benford'sches Gesetz

Erste Ziffer	Häufigkeit nach Benford
1	30,1%
2	17,6%
3	12,5%
4	9,7%
5	7,9%
6	6,7%
7	5,8%
8	5,1%
9	4,6%

Quelle: eigene Darstellung in Anlehnung an: Hill, T.P.: The first-digit phenomenon; erschienen in: American Scientist, Vol. 86, No. 4; Research Triangle Park, North Carolina; 1998; S. 358

[350] Pyle, D.: Data Preparation for Data Mining; Morgan Kaufmann; San Diego; 1999; S. 70
[351] Intuitiv würde man annehmen, dass die Wahrscheinlichkeit für das Auftreten 1/9 = 11,1% betragen würde
[352] Hill, T.P.: The first-digit phenomenon; erschienen in: American Scientist, Vol. 86, No. 4; Research Triangle Park, North Carolina; 1998; S. 358

Attributwerte können auf Einhaltung dieser Verteilung überprüft werden. Voraussetzung ist, dass die zu analysierenden Werte nicht künstlich erzeugt und keine Grenzwerte festgelegt wurden.

Ob ein Attribut künstlich erzeugte Werte enthält, kann neben dem Benford'schen Gesetz überprüft werden, indem die Schrittweite aufsteigend sortierter Werte ermittelt wird. D. Pyle bezeichnet dieses Merkmal als Monotonie und definiert es als Eigenschaft von Attributen mit nach oben unbegrenzten Werten.[353] Im Rahmen dieser Arbeit wird unter dem Begriff Monotonie auch das Auftreten gleichbleibender Schrittweite verstanden. Ist die Schrittweite weitgehend gleich (z.B. Rechnungsnummer, künstlicher Primärschlüssel), so ist das ein Anzeichen für Monotonie.

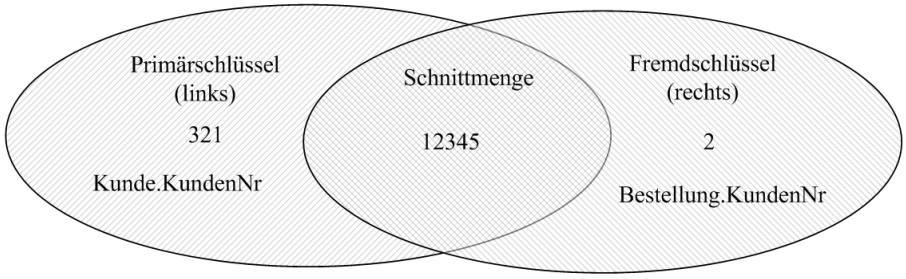
Voraussetzung für ein redundanzfreies Datenbankschema ist ein normalisiertes relationales Datenbankschema. Hierunter versteht man die Aufteilung einer Tabelle in mehrere Tabellen über Normalisierungsalgorithmen, um Redundanzen zu vermeiden. Normalisierung ist eine Entwurfstechnik zur Vermeidung von Redundanz. Das Aufteilen einer Tabelle in weitere Tabellen muss dabei verlustfrei erfolgen, d.h. das Zusammensetzen der aufgeteilten Tabellen über Primär- und Fremdschlüssel muss zur ursprünglichen Tabelle führen. Informationen dürfen dabei nicht verloren gehen. Dazu wird der Begriff der vollen funktionalen Abhängigkeit eingeführt. Danach ist jedes Nicht-Schlüsselattribut voll funktional vom Primärschlüssel abhängig, wenn es durch den gesamten Primärschlüssel eindeutig bestimmt werden kann.[354] Eine Inklusionsabhängigkeit liegt dann vor, wenn ein Attribut A Teilmenge eines anderen Attributes B ist, d.h. alle Werte des Attributes A im Attribut B enthalten sind.[355] Neben referentieller Integrität werden Inklusionsabhängigkeiten auch zur Darstellung von Super-/Subtypenbeziehungen[356] verwendet.

Abhängigkeitsanalysen überprüfen, welche Inklusionsabhängigkeiten existieren und welche Attribute als Primärschlüssel in Frage kommen. Attribute oder Attributkombinationen kann man auf die Eigenschaft Primärschlüsselkandidat testen, indem die Eindeutigkeit der Werte dieser überprüft wird. Referentielle Beziehungen können überprüft werden, indem ermittelt wird, welche Werte zu welchem Prozentsatz in beiden Attributen und nur in einem der beiden

[353] Pyle, D.: Data Preparation for Data Mining; Morgan Kaufmann; San Diego; 1999; S. 71
[354] Cordts, S.: Datenbankkonzepte in der Praxis; Addison-Wesley; München; 2002; S. 82f
[355] Vgl. hierzu: Bauckmann, J.: Efficiently Identifying Inclusion Dependencies in RDBMS; erschienen in: 18. Workshop über Grundlagen von Datenbanken (GI-Workshop),;Wittenberg; 2006
[356] Vgl. hierzu: Cordts, S.: Datenbankkonzepte in der Praxis; Addison-Wesley; München; 2002; S. 49f

vorkommen. Die Werte eines Fremdschlüssels müssen vollständig im Primärschlüssel vorkommen, damit keine Fremdschlüsselverletzung vorliegt. Eine einfache grafische Möglichkeit besteht in der Verwendung von Venn-Diagrammen. Die folgende Abbildung zeigt ein Venn-Diagramm exemplarisch für das Attribut KundenNr der Tabelle Bestellung und das Attribut KundenNr der Tabelle Kunde, um die Beziehung „Ein Kunde hat mehrere Bestellungen“ darzustellen.

Abb. 26: Beispiel für ein Venn-Diagramm zur Analyse von Inklusionsabhängigkeiten



Quelle: eigene Darstellung in Anlehnung an: Mathes, T.; Bange, C.; Keller, P.: Software im Vergleich: Datenqualitätsmanagement, Studie des Business Application Research Center (BARC); Oxygon Verlag; München; 2005; S. 59

Aus dem Venn-Diagramm erkennt man, dass zwei Datensätze nur auf der rechten Seite vorkommen, hier besteht also kein Bezug zur Tabelle Kunde, d.h. zwei Bestellungen verweisen auf nicht vorhandene Kunden. Die linke Seite zeigt 321 Kunden, die noch keine Bestellung vorgenommen haben.

Funktionale Abhängigkeiten können z.B. über Regelinduktion ermittelt werden. Hierzu werden Wenn-Dann-Regeln mit einer hohen Confidence-Kennzahl überprüft. Generell gilt jedoch, dass das Finden von Primärschlüsseln und funktionalen Abhängigkeiten sehr aufwendig ist. T. Dasu und T. Johnson schlagen deshalb einen vierstufigen Algorithmus zum Finden von Primärschlüsseln vor. Dabei werden zunächst diejenigen Attribute eliminiert, die als Primärschlüssel ungeeignet sind (z.B. geringe Eindeutigkeit, viele NULL-Werte, Datentyp Fließkommazahl). Anschließend wird eine Stichprobe aus den Daten gezogen und schrittweise zuerst ein Attribut auf Eindeutigkeit überprüft, dann zwei usw. Die als Primärschlüssel

in Frage kommenden Attribute oder Attributkombinationen werden schließlich mit den gesamten Daten getestet.[357]

Y. Huhtala u.a. partitionieren die Datensätze von Tabellen abhängig von ihren Werten, um funktionale Abhängigkeiten auch für Datenbestände mit einer großen Anzahl Datensätzen zu testen. Der von ihnen beschriebene TANE-Algorithmus funktioniert ähnlich wie die Suche nach Assoziationsregeln.[358] Mehrere Ansätze zur Identifikation von Inklusions- und damit auch referentiellen Abhängigkeiten u.a. auch auf SQL-Ebene beschreibt J. Bauckmann.[359]

Eine Abhängigkeit zwischen zwei numerischen Attributen kann grafisch z.B. über ein Streudiagramm analysiert werden. Der Korrelationskoeffizient gibt Auskunft, ob eine lineare Abhängigkeit zwischen zwei Attributen existiert. Er stellt einen Wert zwischen -1 und +1 dar, wobei die Zahl 0 keinen linearen Zusammenhang, -1 und +1 dagegen einen vollständigen linearen Zusammenhang vermuten lassen. Existiert ein linearer Zusammenhang, so kann über die lineare Regressionsanalyse die Richtung des Zusammenhangs ermittelt werden. Sie kann auch zur Erkennung von Ausreißern verwendet werden.

Aus dem Bereich der Datenbankintegration stammt das Schema Matching. Das sind Verfahren, die semantisch äquivalente Attribute von zwei Schemata automatisch erkennen. Dazu analysiert der sogenannte Matcher die Struktur, die Integritätsbedingungen und eventuell Daten. Schema Matching Verfahren können deshalb auch verwendet werden, um nicht deklarierte Beziehungen zwischen Primär- und Fremdschlüssel zu analysieren.[360] Ein Matcher geht dabei wie folgt vor: Er vergleicht jedes Attribut des Quellschemas mit jedem des Zielschemas und verwendet beim Vergleich Proximitätsmaße, Namen der Attribute und Datentyp usw. Soweit zwei Attribute hinreichend übereinstimmen und einen bestimmten Schwellwert überschreiten, kann von einer Korrespondenz ausgegangen werden.[361] Beim

[357] Dasu, T.; Johnson, T.: Exploratory Data Mining and Data Cleaning; John Wiley & Sons; Hoboken, New Jersey; 2003; S. 174
[358] Vgl. hierzu: Huhtala, Y; Kärkkäinen, J.; Porkka, P.; Toivonen, H.: TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies; erschienen in: The Computer Journal Vol. 42 No. 2; Oxford, United Kingdom ;1999, S. 100-111 und
Huhtala, Y; Kärkkäinen, J.; Porkka, P.; Toivonen, H.: Efficient Discovery of Functional and Approximate Dependencies using Partitions; erschienen in: 14th International Conference on Data Engineering (ICDE'98); IEEE Computer Society Press; Orlando, Florida; 1998; S. 392-401
[359] Bauckmann, J.: Efficiently Identifying Inclusion Dependencies in RDBMS; erschienen in: 18. Workshop über Grundlagen von Datenbanken (GI-Workshop),;Wittenberg; 2006 und
Bauckmann, J.; Leser, U.; Naumann, F.: Efficiently Computing Inclusion Dependencies for Schema Discovery; erschienen in Workshop InterDB (with ICDE06); Atlanta; 2006
[360] Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 144
[361] Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 145f

Vergleich der Namen von Attributen können genau wie bei der Überprüfung der Daten Proximitätsmaße verwendet werden, um die Ähnlichkeit zwischen zwei Attributen zu erkennen. Denkbar ist auch der Einsatz von Approximate String Matching Algorithmen, um die Attributnamen zu vergleichen.

Beim Schema Matching kann unterschieden werden zwischen schemabasierten und instanzbasierten Schema Matching Verfahren. Bei den schemabasierten Verfahren werden die Namen der Attribute und Metadaten zwischen den Attributen verglichen, um eine Korrespondenz zu finden. So würde beispielweise eine Korrespondenz zwischen den beiden Attributen „Titel“ und „Title“ über ein Approximate String Matching erkennbar sein, zwischen „Birthday“ und „Geburtstag“ würde dieses Maß versagen.[362] Instanzbasierte Verfahren verwenden dagegen nicht die Namen und Eigenschaften von Attributen, sondern die Daten selbst. Instanzbasierte Verfahren suchen nach Duplikaten in den Daten oder berechnen statistische Werte (Mittelwert, Varianz, Streuung), um z.B. die Ähnlichkeit zweier Attribute aufzudecken.[363] Ein instanzbasierter Schema Matcher würde bei den Attributen „Birthday“ und „Geburtstag“ ähnliche Eigenschaften (Länge, Zahlen, ähnliche Einträge) analysieren.[364]

[362] Naumann, F.: Informationsintegration - Antrittsvorlesung, Heft 134; Humboldt-Universität zu Berlin; Berlin; 2004; S. 9
[363] Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 146ff
[364] Naumann, F.: Informationsintegration - Antrittsvorlesung, Heft 134; Humboldt-Universität zu Berlin; Berlin; 2004; S. 9

Tab. 19: Mögliche Eingabedaten instanz- und schemabasierter Schema Matcher

	Eigenschaft	Beschreibung
Schema-basiert	Datenlänge	Länge des Datentyps
	Datentyp	CHAR, DATE usw
	Check	Vorhandene CHECK-Constraints
	Primärschlüssel	ja/nein
	Eindeutigkeit	ja/nein
	Fremdschlüssel	ja/nein
	Nullwert	NULL erlaubt
	Wertebereich	Einschränkung des Wertebereichs
	Defaultwert	Vorhandener Standardwert
Instanz-basiert	Minimum	Ermittelt über die Instanzen
	Maximum	Ermittelt über die Instanzen
	Durchschnitt	Ermittelt über die Instanzen
	Kovarianz	Ermittelt über die Instanzen
	Standardabweichung	Ermittelt über die Instanzen

Quelle: eigene Darstellung in Anlehnung an: Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 151

Hybride Verfahren kombinieren schema- und instanzbasierte Schema Matching Verfahren miteinander und erzielen normalerweise effizientere Ergebnisse.[365]

Der von A. Bilke beschriebene Schema Matcher DUMAS (DUplicate-based MAtching of Schemata) erkennt Duplikate zwischen unterschiedlichen Tabellen und ermittelt auf dieser Grundlage Korrespondenzen zwischen Attributen.[366]

Bei der Analyse des Data Tracking werden Daten über einen oder mehrere Geschäftsprozesse (Information Chain) analysiert und die Veränderungen gemessen.[367] Dieses Verfahren ist zwar sehr aufwendig, bietet aber die Möglichkeit, die Ursachen eines Datenqualitätsproblems genau zu erforschen. Über das Data Tracking können zudem Zeiten von Geschäftsprozess-zyklen gemessen werden.

[365] Naumann, F.: Informationsintegration - Antrittsvorlesung, Heft 134; Humboldt-Universität zu Berlin; Berlin; 2004; S. 9f
[366] Bilke, A.: Duplicate-based Schema Matching, Dissertation; Technische Universität Berlin, Fakultät Elektrotechnik und Informatik; Berlin; 2007; S. 16
[367] Redman, T.: Data Quality for the Information Age; Artech House; Norwood; 1996; S. 211

3.4.3 Regeln

Geschäftsprozesse werden durch Geschäftsregeln bestimmt, die von einem Domänenexperten ermittelt und erfasst werden. Die Regeln sollten in bestimmten Zeitabständen auf Verletzung überprüft werden. Diese Anwendung von Regeln bezeichnen C. Batini und M. Scannapieco als Error Localization oder Error Detection.[368] Danach werden semantische Regeln auf einen Datenbestand angewendet und Datensätze, die den Regeln nicht entsprechen, als falsch identifiziert.

Noch sinnvoller ist es natürlich, Geschäftsregeln in ein RDBMS zu integrieren, um eine Regelverletzung zu protokollieren bzw. von vornherein zu vermeiden. Dadurch könnte die Qualität von Daten sukzessive in einem evolutionären Prozess verbessert werden.

Verletzungen von Regeln sollten protokolliert werden können, um damit z.B. eine Benutzerstatistik zu erstellen, auf deren Basis Geschäftsprozesse verbessert werden können.

Regeln dürfen als Gesamtregelwerk nicht redundant oder inkonsistent sein:

Beispiel für inkonsistente Regeln:

Bestellung < 2.000 Euro

und

Kundenkarte = Gold und Bestellung < 5.000 Euro

Beispiel für redundante Regeln:

Bestellung < 2.000 Euro

und

Kundenkarte = Standard und Bestellung < 2.000 Euro

Außerdem sollte es möglich sein, dass Regeln auch auf Erfahrungswerten basieren, wenn einem Domänenexperten z.B. bekannt ist, dass pro Monat eine Tabelle um etwa 1.000

[368] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 219

Datensätze wächst.[369] Eine Berücksichtigung der Größe des Datenvolumens kann auch Rückschlüsse auf vorhandene Probleme liefern.

Auch Regeln auf Grundlage von Häufigkeiten, wenn man z.B. deren prozentuale Anteile kennt, kann durchaus sinnvoll sein. Nimmt man z.B. als Grundlage seines Kundenbestandes die demografischen Daten in Deutschland, so ist das Verhältnis der Anzahl von Männern und Frauen etwa 51% zu 49%. Eine Geschäftsregel könnte diesen Zusammenhang berücksichtigen.[370]

Die Formulierung der Regeln sollte in Anlehnung an eine Business Rule Engine von Fachexperten formuliert werden und deshalb intuitiv erfasst werden können. Eine andere Möglichkeit wäre die Formulierung von Regeln in SQL mit benutzerdefinierten Funktionen. Die folgende SQL-Anweisung gibt beispielhaft die Anzahl der Formatverletzungen bei der Datumseingabe an:

```
SELECT COUNT(*)
FROM Adressen
WHERE PATTERN(Geburtstag, 'dd.mm.yyyy') <> 'dd.mm.yyyy'
```

3.5 Verifizieren und Verbessern

3.5.1 Standardisieren

Zum Standardisieren gehören das Vereinheitlichen oder Transformieren der Daten und das Parsing oder Restrukturieren. Voraussetzung für das Korrigieren und Anreichern ist die Bereitstellung der Daten in einem einheitlichen, standardisierten Format, wobei alle Spalten weitgehend atomare Werte enthalten sollten. Zur Vereinheitlichung der Daten gehört die Anpassung von Werten, so dass sie alle die gleiche semantische Bedeutung und eine einheitliche Repräsentation haben. Es werden einheitliche Regeln festgelegt (z.B. m, w, NULL für die Spalte Geschlecht), wie Daten repräsentiert werden sollen. Eine Spalte mit der

[369] Winter, M.; Helfert, M.; Herrmann, C.: Das metadatenbasierte Datenqualitätssystem der Credit Suisse; erschienen in: von Maur, E.; Winter, R. (Hrsg.): Vom Data Warehouse zum Corporate Knowledge Center - Proceedings der Data Warehousing 2002; Physica-Verlag; Heidelberg; 2002; S. 168

[370] Vgl. hierzu: Helfert, M.: Proaktives Datenqualitätsmanagement in Data-Warehouse-Systemen, Dissertation; logos-Verlag; Berlin; 2002; S. 170

Firmenbezeichnung sollte z.B. die Werte „Volkswagen AG“, „VW AG“, „Volkswagen Aktiengesellschaft“ in einen vereinheitlichen Standardwert (z.B. „Volkswagen AG“) transformieren, da alle Begriffe semantisch dasselbe reale Objekt bezeichnen. Vereinheitlichen kann auf Grundlage von benutzerdefinierten Regeln, lokalen bzw. unternehmensweiten Vereinheitlichungsregeln oder aber auf Basis von Industriestandards durchgeführt werden. Regeln zur Vereinheitlichung der Daten korrigieren zwar keine Fehler, erleichtern aber die Bearbeitung und die Analyse der Daten.[371] Zum Standardisieren gehören neben dem Normieren von Abkürzungen und Begriffen auch das Entfernen von Rausch- bzw. Stoppwörtern und die Korrektur von Rechtschreibfehlern.

Um Abkürzungen und Akronyme anzupassen, kann z.B. eine Nachschlagetabelle verwendet werden. Nachschlagetabellen können zwar sehr spezifisch an das jeweilige Geschäftsumfeld angepasst werden, haben jedoch den Nachteil eines erheblichen Wartungsaufwandes, gerade wenn Bezeichnungen auch in verschiedenen Sprachen vorliegen. Zudem sind vor Erstellung solcher Nachschlagetabellen die Daten detailliert zu analysieren, um Regeln in der Darstellung zu finden. Zum Beispiel ist nicht eindeutig, ob mit „VW AG“ auch wirklich die „Volkswagen AG“ in jedem Datenbestand gemeint ist. Hierzu ist eine Analyse der Daten z.B. anhand von Häufigkeiten sinnvoll, um dann nach dem häufigsten auftretenden Wert zu standardisieren. Andere Möglichkeiten zur Vereinheitlichung sind z.B. die Vorgabe eines Datenmusters oder die Anwendung von Verfahren zur „Named Entity Recognition“. Zur Vereinheitlichung gehört auch die allgemeine Homogenisierung von Zeichenketten (Groß-/Kleinschreibung, Camel-Case-Schreibweise).

Datenwerte können über Konvertierungsfunktionen von einer Maßeinheit in eine andere transformiert werden (Längenangaben, Temperaturen, finanzielle Daten)[372] und NULL durch plausible Werte ersetzt werden.

Parsing erkennt multiple Werte in Attributen, die die erste Normalform verletzen und isoliert diese in atomare Attribute (z.B. das Attribut Adresse in die Attribute Anrede, Vorname, Nachname, Strasse, Hausnummer, Plz, Ort, Bundesland, Staat). Dazu muss die Semantik der einzelnen Teilzeichenketten in der Regel über Nachschlagetabellen erkannt werden. Neben der ausschließlichen Verwendung von Nachschlagetabellen gibt es andere Ansätze aus den Bereichen des Natural Language Programming (NLP), des Information Retrievals und des

[371] Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 326
[372] Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 327

Text Minings, die auf Basis des Datenbestandes die Semantik von Wörtern mehr oder weniger erlernen.[373] Ein häufig verwendeter Ansatz hierzu sind Hidden Markov Modelle (HMM). Dazu werden zunächst die Attributwerte in eine Liste von Wörtern aufgetrennt. Diese wird dann über das Tagging[374] markiert, wobei die Marken die Semantik der Worte festlegen. Die Wörter dieser markierten Liste werden daraufhin über das HMM den Ausgabeattributen zugewiesen, die am wahrscheinlichsten erscheinen. So kann z.B. das Wort „Franz" sowohl Nachname wie auch Vorname sein. Über das Modell wird entschieden, welche Marke (Nachname oder Vorname) wahrscheinlicher ist.[375] Markov Modelle werden dann verwendet, wenn die Wahrscheinlichkeit von linearen Attributsequenzen, deren einzelne Werte nicht unabhängig voneinander sind, vorhergesagt werden soll.[376]

3.5.2 Anreichern

Fügt man einem Datenbestand weitere Daten hinzu, so spricht man von Datenanreicherung. Eine Datenanreicherung ist z.B. sinnvoll, um einen Datenbestand um demografische Daten bei Kunden zu ergänzen, um damit die Duplikaterkennung zu verbessern oder aber neue Geschäftsregeln aus dem Datenbestand zu analysieren. Neben demografischen Daten können auch standardisierte Produktcodes wie EPC (Elektronischer Produktcode), EAN (European bzw. International Article Number) oder UPC (Universal Product Code) zur Anreicherung sinnvoll sein. Einen Spezialfall stellt das Hinzufügen von Längen- und Breitengraden bei Ortsangaben dar. Bei der Duplikaterkennung kann dadurch z.B. ermittelt werden, ob für Personen mit generell gleichen Daten aber unterschiedlicher Ortsangabe eventuell ein Ortswechsel stattgefunden hat, da über das Geocoding die Entfernung zwischen zwei Orten berechnet werden kann.

Auch durch die Verwendung eines Algorithmus in Form z.B. eines Klassifizierers können Daten angereichert werden, ohne dass dies über Nachschlagetabellen erfolgt. Dabei wird der Algorithmus anhand eines Datenbestandes trainiert, um den Wert einer Spalte vorherzusagen.

---

[373] Vgl. hierzu: Wen, Y.: Text Mining Using HMM and PPM, University of Waikato, Hamilton, 2001
[374] Für das Tagging verwendet dieser Ansatz allerdings auch Nachschlagetabellen
[375] Christen, P. u.a.: Probabilistic Name and Address Cleaning and Standardisation, The Australasian Data Mining Workshop, o.A., 2002, S. 3f
[376] Manning, C. D., Schütze, H.: Foundations of Statistical Natural Language Processing; The MIT Press; Cambridge, Massachusetts; 2003; S. 318f

3.5.3 Duplikate

3.5.3.1 Einführung

Unter Duplikaten versteht man mehrfach gespeicherte Datensätze, die sich auf dasselbe Objekt in der realen Welt beziehen.[377] Der Vorgang des Erkennens von Duplikaten wird als Object Identification, Record Linkage, Record Matching oder Record Resolution bezeichnet. Record Linkage bezieht sich dabei eher auf einfache strukturierte Daten, wie Dateien oder Tabellen. Beschränkt sich die Erkennung von Duplikaten auf eine einzige Datei, wird auch der Begriff Deduplication oder Duplicate Identification verwendet. Object Identification berücksichtigt auch komplexere Strukturen. [378]

Wenn jeder Datensatz in einer Datenbanktabelle einen eindeutigen und fehlerfreien Identifikationscode hätte, könnten Duplikate durch Vergleich dieses Codes erkannt werden.[379] In einer relationalen Datenbank wäre dann ein einfacher Equi-Join ausreichend, um die Duplikate zusammenzuführen. Da Daten jedoch nicht fehlerfrei sind, benötigt man Vergleichsalgorithmen, die zwischen zwei Werten Unschärfen erkennen. Damit ist auch bei der Integration einer oder mehrerer Datenbestände das Problem der semantischen gegenüber der strukturellen Heterogenität als höher einzustufen.[380]

Der Schwerpunkt bei der Duplikaterkennung kann entweder in der Einfachheit der Anwendung des Verfahrens oder aber in der Präzision der Erkennung von Duplikaten liegen. Abhängig davon können unterschiedliche Methoden angewendet werden. So ist z.B. bei einem Mailing die Höhe der Erkennungsrate nicht so entscheidend wie im medizinischen Forschungsbereich, zum Herausfiltern gleicher Patienten. Bei einem Mailing kann es durchaus ausreichen, über einen einfachen Matchcode Duplikate zu finden. Je höher und präziser die Erkennungsrate sein soll, umso höher sind normalerweise auch die Kosten. So

[377] Ein Spezialfall der Duplikaterkennung stellt das sog. Householding dar, bei dem Datensätze gesucht werden, die sich auf den gleichen Haushalt beziehen (Vgl. hierzu: Helfert, M.: Proaktives Datenqualitätsmanagement in Data-Warehouse-Systemen, Dissertation; logos-Verlag; Berlin; 2002; S. 51)
[378] Vgl. hierzu: Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 100
[379] Cochinwala, M.; Dalal, S.; Elmagarmid, A. K.; Verykios, V. S.: Record matching: Past, present and future; Technical Report TR-01-013; Purdue University, Department of Computer Sciences; 2001; S. 4
[380] Vgl. hierzu: Ziegler, P.; Dittrich, K.R.: Three Decades of Data Integration - All Problems solved?; erschienen in: Jacquart, R. (Eds.): 18th IFIP World Computer Congress (WCC 2004), Volume 12, Building the Information Society, August 22-27; Kluwer; Toulouse, France; S. 9

kann es durchaus sinnvoll sein, eine bestimmte Anzahl nicht erkannter Duplikate zugunsten einer einfachen Duplikatklassifikation zuzulassen, wenn dadurch die Gesamtkosten niedriger sind.

Die Bereinigung von Duplikaten setzt sich aus drei Schritten zusammen:

1) Reduktion des Suchraums
2) Erkennung der Duplikate (Searching, Duplikaterkennung)
3) Zusammenführen der Duplikate (Merging, Duplikat- oder Datenfusion)

Im Folgenden werden die einzelnen Schritte und die bekanntesten existierenden Verfahren hierzu beschrieben. Zur Erläuterung werden die Daten aus Tabelle 20 verwendet.

Tab. 20: Beispieldaten zur Bereinigung von Duplikaten

Satznr.	**Nachname**	**Vorname**	**Plz**	**Ort**
1	Muster	Burkhard	25746	Heide
2	Muster	Burkhard	25746	Heide
3	Mustr	Bukrhard	25746	Heide
4	Muster	Burghart	25746	Heide
5	Müller	Burkhard	22525	Hamburg
6	Duwenga	Batanayi	25746	Heide
7	Duwenga	Batanayi	22525	Hamburg

Quelle: eigene Darstellung

3.5.3.2 Reduktion des Suchraums

Um in einer Datenbanktabelle alle Duplikate zu erkennen, muss normalerweise jeder Datensatz mit jedem anderen verglichen werden. Für das Beispiel aus Tabelle 20 sind danach 21 unterschiedliche Vergleiche notwendig. Die Anzahl der Vergleiche berechnet sich nach folgender Formel[381]:

[381] Elfeky, M G.; Elmagarmid, A. K.; Ghanem, T. M.: Towards Quality Assurance of Government Databases: A Record Linkage Web Service; Demo at DG.O'2002, The 3rd National Conference on Digital Government Research; Los Angeles, California; 2002; S. 2

$$\frac{n \cdot (n-1)}{2} = \frac{n^2 - n}{2} = \frac{7^2 - 7}{2} = 21$$

mit

n Anzahl der Datensätze

Bei Tabellen mit einer geringen Anzahl an Datensätzen ist der Berechnungsaufwand noch praktikabel. Bereits bei 10.000 Datensätzen sind nach diesem Verfahren allerdings schon etwa 50 Millionen Datensatzvergleiche notwendig. Um den Suchraum von möglichen Duplikaten zu reduzieren, versucht man nun, Datensätze möglichst so zu gruppieren, dass die Wahrscheinlichkeit vorhandener Duplikate in den einzelnen Gruppen hoch ist und offensichtliche Nicht-Duplikate ausgefiltert[382] werden.

Zur Reduktion des Suchraumes werden die folgenden Verfahren beschrieben:

- Blocking
- Sorted Neighboorhood Methode (SNM) bzw. Windowing
- N-Gram basiertes Blocking
- Canopy Clustering
- Suffix Array basiertes Blocking

Das bekannteste Verfahren zur Partitionierung von Datensätzen ist das Blocking. Hierbei werden ein oder mehrere Attribute zu Blockingvariablen zusammengefasst. Nur Datensätze mit dem gleichen Wert der Blockingvariablen werden anschließend miteinander verglichen. Für die obigen Beispieldaten verringern sich die Vergleiche beim Blocking auf insgesamt 11, wenn als Blockingvariable die Postleitzahl verwendet wird. Die Postleitzahl 25746 tritt fünfmal und die 22525 zweimal auf. Damit ergibt sich folgende Anzahl an Datensatzvergleichen:

$$\frac{5^2 - 5}{2} + \frac{2^2 - 2}{2} = 10 + 1 = 11$$

[382] Goiser, K.; Christen, P.: Towards Automated Record Linkage; erschienen in: Christen, P.; Kennedy, P. J.; Jiuyong, L.; Simoff, S. J.; Williams, G. J. (Hrsg.): Data Mining and Analytics 2006, Proceedings of the Fifth Australasian Data Mining Conference, Conferences in Research and Practice in Information Technology (CRPIT), Vol. 61.; ACS Press; Sydney; S. 24

Das Blocking setzt allerdings voraus, dass Attribute, die zum Reduzieren des Suchraums verwendet werden, weitgehend fehlerfrei sind und den Datenbestand in eine ausreichend hohe Anzahl von Gruppen unterteilen. Die Auswahl der Blockingvariablen, auch als Schlüssel bezeichnet, ist dabei abhängig von der Anwendungsdomäne und wird normalerweise manuell von einem Domänenexperten vorgenommen[383], der die Charakteristiken der Daten und Fehler in den Schlüsseldaten berücksichtigt. Zudem müssen die Schlüsseldaten nicht zwangsläufig aus den Werten selbst bestehen, sondern können auch aus den Daten abgeleitet werden (z.B. Soundexcode für Nachname, nur die Vorwahl der Telefonnummer).[384] Leicht änderbare Werte wie Adressen sind zum Blocking ungeeignet, stabile Attribute wie das Geburtsjahr danach eher geeignet.[385] Bei der Auswahl der Blockingvariablen können z.B. Häufigkeitsverteilungen helfen, einen geeigneten Schlüssel zu finden. Das Ergebnis der Duplikaterkennung ist somit abhängig von der Qualität des Schlüssels zum Blocking.

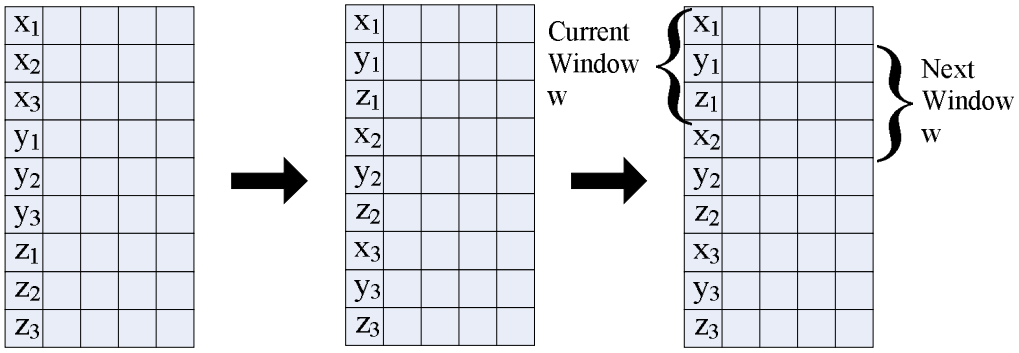
Neben der Fehlerfreiheit und Stabilität der Blockingvariable spielt auch die Häufigkeitsverteilung der Blockingvariable eine Rolle. Wird der Datenbestand durch das Blocking in einen großen und mehrere kleine Blöcke eingeteilt, so dominiert der große Block die Anzahl der Vergleiche. Idealerweise sollte das Blocking den Datenbestand in Blöcke gleicher Größe aufteilen.[386] Das obige Beispiel aus Tabelle 20 hat z.B. einen großen Block mit 5 und einen relativ kleinen mit nur 2 Datensätzen. Bei einer Gleichverteilung von 4 zu 3 Datensätzen reduziert sich die Anzahl der Vergleiche auf insgesamt 9. Bei einem Datenbestand von 10.000 Datensätzen und einer Gleichverteilung von jeweils 100 Datensätzen je Block sind etwa 500.000 Vergleiche notwendig. Dagegen liegt die Anzahl der Vergleiche bei einer Verteilung von 99 Blöcken mit nur einem Datensatz und einem Block von 9.901 Datensätzen bei über 49 Millionen.

Beim zweiten Verfahren, der Sorted Neighborhood Method (SNM), werden die Datensätze ebenfalls nach einer Blockingvariable bzw. einem Schlüssel sortiert und ein Fenster fixer

[383] Christen, P.: Improving data linkage and deduplication quality through nearest-neighbour based blocking; erschienen in: Proceedings of thirteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07); San Jose, California; 2007; o. S.
[384] Vgl. hierzu: Naumann, F.: Informationsintegration - Antrittsvorlesung, Heft 134; Humboldt-Universität zu Berlin; Berlin; 2004; S. 14
[385] Gill, L.: The Oxford Medical Record Linkage System, Record Linkage Techniques; erschienen in: Proceedings of an International Workshop and Exposition; Arlington; 1997; S. 18
[386] Christen, P.: Improving data linkage and deduplication quality through nearest-neighbour based blocking; erschienen in: Proceedings of thirteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07); San Jose, California; 2007; o. S.

Größe über die Datensätze gelegt. Darauf werden alle Datensätze innerhalb des Fensters verglichen und das Fenster jeweils um einen Datensatz weitergeschoben, wie in Abb. 27 zu erkennen ist.[387]

Abb. 27: Sorted Neighborhood Method (SNM)



Quelle: Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 115

Die grundlegende Idee des Sorted Neigborhood stammt aus dem Jahr 1983 von D. Bitton und D.J. DeWitt. Exakte Duplikate werden identifiziert, indem die Tabelle sortiert und benachbarte Datensätze miteinander verglichen werden.[388] Die Bezeichnung Sorted-Neighborhood Methode (SNM) oder Merge-Purge Methode stammt jedoch von M.A. Hernandez und S.J. Stolfo und beinhaltet neben dem Verfahren zur Reduktion des Suchraumes auch ein Verfahren zur Duplikaterkennung.[389] Zur Reduktion des Suchraumes wird, wie bereits erwähnt, zunächst ein Schlüssel aus einem oder mehreren Attributwerten erzeugt. Bei einer Tabelle mit Kundendaten könnte sich der Schlüssel z.B. aus einem 3-stelligen phonetischen Soundexcode des Nachnamens und dem Geburtsjahr zusammensetzen. Die Datensätze werden nach dem erzeugten Schlüssel sortiert und schließlich ein Fenster fixer Größe über die Datensätze bewegt. Dabei werden nur die Datensätze innerhalb des Fensters paarweise untereinander verglichen werden.[390] Wird das Fenster um einen Datensatz weitergeschoben, so werden alle anderen Datensätze im Fenster mit diesem verglichen.

[387] Cochinwala, M.; Dalal, S.; Elmagarmid, A. K.; Verykios, V. S.: Record matching: Past, present and future; Technical Report TR-01-013; Purdue University, Department of Computer Sciences; 2001; S. 8
[388] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 113
[389] Hernandez, M.A.; Stolfo, S.J.: The Merge/Purge Problem for Large Databases; Proceedings of the 1995 ACM SIGMOD international conference on Management of data; San Jose, California; 1995; S. 127-138
[390] Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 340f

Wird in dem obigen Beispiel aus Tabelle 20 die Postleitzahl als Schlüssel ausgewählt und die Fenstergröße auf 2 gesetzt, verringert sich die Anzahl der Vergleiche auf insgesamt 6. Die Anzahl der Vergleiche berechnet sich nach folgender Formel:

$$\frac{(w^2 - w)}{2} + (n - w) \cdot (w - 1) = \frac{(2^2 - 2)}{2} + (7 - 2) \cdot (2 - 1) = 1 + 5 = 6$$

mit

n Anzahl der Datensätze

w Fenstergröße

Bei 10.000 Datensätzen und einer Fenstergröße von 20 müssen nur noch etwa 190.000 Vergleiche durchgeführt werden (im Vergleich zu etwa 50 Millionen ohne SNM).

Liegen zwei Duplikate nach dem Sortieren weit auseinander, ist es genau wie beim Blocking unwahrscheinlich, dass sie im Fenster erscheinen, es sei denn man vergrößert die Fenstergröße[391] (das wiederum hätte allerdings nachteilige Auswirkungen auf die Anzahl der Vergleiche). Neben der Auswahl der Proximitätsmaße hängt das Ergebnis der Duplikaterkennung auch hier von der Wahl eines geeigneten Schlüssels und der Größe des Fensters ab. Untersuchungen haben gezeigt, dass bei geeigneter Schlüsselwahl eine Fenstergröße von 20 ausreicht und die Wahl des Schlüssels für die Qualität der Erkennung entscheidend ist. Vor allem das Attribut, das die ersten Zeichen des Schlüssels bildet und damit die Sortierung maßgeblich vorgibt, ist wichtig. SNM wurde deshalb um die Verwendung mehrerer Schlüssel und mehrerer Durchläufe im Multipass-SNM weiterentwickelt. Dabei zeigt sich, dass selbst bei kleinen Fenstergrößen gute Ergebnisse erzielt wurden.[392] Durchläufe mit geringer Fenstergröße wiederum erhöhen die Performanz, da damit die Anzahl der Vergleiche sinkt. Die Korrektheit gegenüber der ursprünglichen SNM ist dabei drastisch höher.[393]

Eine andere Variante der SNM ist die inkrementelle SNM: Ein erster Durchlauf erfolgt nach dem SNM, danach werden nur noch repräsentative Datensätze für folgende Vergleiche

[391] Vgl. hierzu: Low, W.L.; Lee, M.L.; Ling, T.W.: A Knowledge-Based Approach for Duplicate Elimination in Data Cleaning; erschienen in: Information Systems, Vol. 26, Issue 8 (December 2001); Elsevier Science; Oxford, UK; S. 589

[392] Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 342

[393] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 115

verwendet.[394] Dieser Ansatz ist für Datenbestände geeignet, bei denen kontinuierlich Datensätze neu hinzugefügt werden.

Beim N-Gram basierten Blocking werden, wie der Name schon ausdrückt, aus der Blockingvariable N-Grams gebildet (siehe hierzu Abschnitt 3.2.3 Tokenizer). Abhängig von einem Schwellwert wird eine bestimmte Anzahl an N-Grams kombiniert. Hierzu ein Beispiel: Es sollen N-Grams der Größe 2 gebildet werden und der Schwellwert liegt bei 0,8. Aus der Zeichenkette „hans“ werden die 2-Grams 'ha', 'an', 'ns' gebildet. Die Anzahl 2-Grams wird nun mit dem Schwellwert multipliziert und ergibt die Anzahl zu bildender Kombinationen: $Round(3 \cdot 0{,}8) = 2$. Nun werden aus den 2-Grams alle 2er Kombinationen gebildet und zu einer Zeichenkette als Blockingvariable konkateniert:

'ha', 'an'
'ha', 'ns'
'an', 'ns'

Zusätzlich werden die ursprünglichen 2-Grams zu einer Zeichenkette zusammengefasst. Damit ergeben sich für diesen Datensatz als Blockingvariablen 'haanns', 'haan', 'hans', 'anns'.[395] Im Gegensatz zu den zuvor beschriebenen Verfahren wird beim N-Gram basierten Blocking ein Datensatz mehreren Blöcken zugeteilt.

Beim Canopy Clustering wird ein kostengünstiges Ähnlichkeitsmaß verwendet, um über ein Clusterverfahren alle nach dem Ähnlichkeitsmaß zusammengehörenden Datensätze in Cluster, als Canopy („Überdachung“, „Vordach“) bezeichnet, zusammenzufassen.[396]

Beim Suffix Array basierten Blocking wird, ähnlich wie beim N-Gram basierten Blocking, jeder Datensatz mehreren Blöcken zugeordnet. Dazu werden der Wert der Blockingvariable und alle Suffixe bis zu einer bestimmten Größe als Werte der Blockingvariablen verwendet.

[394] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 115f
[395] Vgl. hierzu: Christen, P.: Improving data linkage and deduplication quality through nearest-neighbour based blocking; erschienen in: Proceedings of thirteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07); San Jose, California; 2007; o. S.
[396] Baxter, R.; Christen, P.; Churches, T.: A Comparison of Fast Blocking Methods for Record Linkage; erschienen in: Proceedings of the Workshop on Data Cleaning, Record Linkage and Object Consolidation at the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; Washington DC; 2003; o. S.

Bei einer minimalen Suffixlänge von 3 bildet man aus der Zeichenkette „muster“ die Schlüssel „muster“, „uster“, „ster“ und „ter“.

3.5.3.3 Duplikate erkennen

Sollen Paare von Datensätzen verglichen werden, muss entschieden werden, ob diese das gleiche reale Objekt repräsentieren. Es muss also festgelegt werden, ob ein Paar übereinstimmt (Match), nicht übereinstimmt (Unmatch) oder ob es vielleicht übereinstimmt (Possible) und eine Entscheidung manuell erfolgen muss. Die Erkennung von Duplikaten besteht also aus zwei Schritten: Dem Vergleich eines Datensatzpaares und der Klassifikation[397] in eine der drei Klassen Match, Unmatch oder Possible. Dazu werden die einzelnen Attribute des Datensatzpaares verglichen und Indikatoren für die Übereinstimmung oder Nicht-Übereinstimmung ermittelt. Diese Indikatoren müssen dann zu einem Gesamtindikator oder einer Bewertungskennzahl (rating score) zusammengefasst werden. Im einfachsten Fall können diese addiert bzw. der Mittelwert gebildet werden. Abhängig von der Bedeutung des Attributs oder von der Häufigkeit des Auftretens eines Wertes im Datenbestand oder einem Referenzdatenbestand kann auch eine Gewichtung erfolgen.[398]

Allgemein kann bei den Erkennungsmethoden zwischen probabilistischen und deterministischen (im engeren Sinne regelbasierten) Erkennungsverfahren unterschieden werden. Probabilistische verwenden im Gegensatz zu deterministischen Verfahren Wahrscheinlichkeiten zur Berechnung eines Übereinstimmungswertes zwischen zwei Datensätzen und berücksichtigen damit die Wahrscheinlichkeit des Auftretens eines Wertes.

Das bekannteste probabilistische Verfahren ist das Record Linkage Verfahren nach Fellegi und Sunter. Die Bezeichnung Record Linkage wurde erstmals von H.L. Dunn im „American Journal of Public Health“ erwähnt.[399] Die Grundlagen zum Verfahren wurden 1962 von

[397] Goiser, K.; Christen, P.: Towards Automated Record Linkage; erschienen in: Christen, P.; Kennedy, P. J.; Jiuyong, L.; Simoff, S. J.; Williams, G. J. (Hrsg.): Data Mining and Analytics 2006, Proceedings of the Fifth Australasian Data Mining Conference, Conferences in Research and Practice in Information Technology (CRPIT), Vol. 61.; ACS Press; Sydney; S. 23
[398] Gu, L.; Baxter, R.A.; Vickers, D.; Rainsford, C.: Record Linkage: Current Practice and Future Directions, Technical Report 03/83; CSIRO Mathematical and Information Sciences; Canberra, Australia; 2003; S. 13
[399] Dunn, H.L.: Record Linkage; erschienen in: American Journal of Public Health, Vol. 36; New York; 1946; S. 1412–1416.

H. Newcombe[400] gelegt, indem er Entscheidungsregeln zum Ableiten von übereinstimmenden (Matches) und nicht-übereinstimmenden Datensätzen (Unmatches) einführte, um Duplikate zu erkennen. 1969 wurden schließlich die formalen mathematischen Grundlagen von I. Fellegi und A. Sunter beschrieben, die auf der Wahrscheinlichkeitstheorie basieren.[401]

Um zu überprüfen, ob zwei Datensätze Duplikate sind, wird jedes Attribut beider Datensätze miteinander verglichen. Jedes zu überprüfende Attribut geht nach H.B. Newcombe mit einem Einzelgewicht in ein Gesamtgewicht ein, das je nach Höhe bestimmt, ob zwei Datensätze übereinstimmen (Matches), nicht übereinstimmen (Unmatches) oder vielleicht übereinstimmen (Possible Match). Die Gewichte basieren auf Häufigkeitsverhältnissen zwischen Matches und Unmatches eines Datenbestandes, für den diese z.B. manuell klassifiziert wurden. Ein Attribut mit einem hohen Wert als Häufigkeitsverhältnis ist dabei besser geeignet, gleiche Datensätze zu identifizieren als ein Attribut mit einem niedrigen Häufigkeitsverhältnis zwischen Matches und Unmatches.

Die Berechnung des Einzelgewichtes bei Übereinstimmung eines Attributes als Häufigkeitsverhältnis erfolgt durch Logarithmieren zur Basis 2:[402]

$$Gewicht_{agree} = \log_2\left(\frac{Matches}{Unmatches}\right)$$

Wurden z.B. bei einem repräsentativen Datenbestand die Matches und Unmatches manuell ermittelt und hat sich herausgestellt, dass der Nachname in der Gruppe der Matches in 93% und in der Gruppe der Unmatches in 2% übereinstimmt, ergibt sich für das Häufigkeitsverhältnis ein Wert von 0,93/0,02 und ein Übereinstimmungsgewicht von 5,54. Beim Geschlecht könnte sich bei der Gruppe der Matches eine Übereinstimmung von 95% und bei der der Gruppe der Unmatches von 50% ergeben. Das Übereinstimmungsgewicht liegt dann nur bei 0,93. Diese Differenz im Gewicht ist auch intuitiv nachvollziehbar, da das Attribut Nachname zur Unterscheidung zwischen Personen besser geeignet ist als ausschließlich das Geschlecht.

[400] Newcombe, H. B.; Kennedy, J. M.: Record linkage: making maximum use of the discriminating power of identifying information; Communications of the ACM archive Volume 5 , Issue 11; New York; 1962; S. 563-566

[401] Fellegi, I.; Sunter, A.: A Theory for Record Linkage; Journal of the American Statistical Association Vol. 64 No. 328, American Statistical Assocation, o.A., 1969, S. 1183-1210

[402] Gill, L.: Methods for Automatic Record Matching and Linkage and their Use in National Statistics, National Statistics Methodological, Series No. 25; National Statistics; London; 2001; S. 64

Generell ist festzustellen, dass Attribute mit vielen Ausprägungen in der Regel auch ein höheres Gewicht erhalten als die mit wenigen unterschiedlichen Ausprägungen.

Analog wird auch ein Gewicht bei Nicht-Übereinstimmung errechnet. Dieses ergibt sich aus folgender Formel:

$$Gewicht_{disagree} = \log_2\left(\frac{1 - Matches}{1 - Unmatches}\right)$$

Für das Beispiel mit den Attributen Nachname und Geschlecht ergibt sich ein Gewicht bei Nicht-Übereinstimmung der Wert von -3,81 beim Nachnamen und -3,32 beim Geschlecht.[403]

Für jedes Attribut der beiden zu vergleichenden Datensätze werden nun die Übereinstimmungs- und Nicht-Übereinstimmungsgewichte berechnet und addiert. Abhängig von der Höhe des Gesamtgewichtes wird das Datensatzpaar als Match, Unmatch oder Possible Match klassifiziert. Zur Entscheidung werden zwei benutzerdefinierte Schwellwerte benötigt, ein unterer und ein oberer. Datensatzpaare mit einem Gesamtgewicht oberhalb des oberen Schwellwertes werden der Gruppe der Matches und Paare mit einem Gewicht unterhalb des unteren Schwellwertes der Gruppe Unmatches zugeordnet. Gesamtgewichte, die zwischen dem unteren und oberen Schwellwert liegen, stimmen eventuell überein und sollten manuell klassifiziert werden.

Ein Nachteil beim Record Linkage Verfahren ist die manuelle Klassifizierung eines repräsentativen Datenbestandes in die Gruppen Matches und Unmatches, um die Übereinstimmungs- und Nicht-Übereinstimmungsgewichte zu berechnen. Diese Gewichte können jedoch auch auf Basis des zu klassifizierenden Datenbestandes approximativ mit folgenden Formeln berechnet werden:[404]

[403] Vgl. hierzu: Cordts, S.; Müller, B.: Dublettenbereinigung nach dem Record Linkage Algorithmus; erschienen in: Cremers, A. u.a. (Hrsg.): Beiträge der 35. Jahrestagung der Gesellschaft für Informatik e.V., Informatik 2005 - Informatik LIVE!, Band 2, Bonner Köllen Verlag, Bonn, 2005; S. 431

[404] Gill, L.: Methods for Automatic Record Matching and Linkage and their Use in National Statistics, National Statistics Methodological, Series No. 25; National Statistics; London; 2001; S. 63f
und siehe Quellcodedatei Comparison.py des Open Source Projekts febrl (Freely Extensible Biomedical Record Linkage) unter http://sourceforge.net/projects/febrl, Version 0.2.2

$$Gewicht_{agreeAusprägung} = \log_2\left(\frac{1}{p}\right)$$

$$Gewicht_{disagreeAusprägung} = \log_2\left(\frac{1-p}{1-p^2}\right)$$

mit

p Anteil der Ausprägung an den Gesamtausprägungen

Hierzu ein Beispiel: Ein Kundendatenbestand mit 10.000 Datensätzen hat für das Attribut Geschlecht 6.000mal den Wert „männlich“ und 4.000mal den Wert „weiblich“. Das Übereinstimmungsgewicht für die Ausprägung „männlich“ beträgt damit 0,74 und das Nicht-Übereinstimmungsgewicht -0,68.

Ein weiterer Nachteil beim Record Linkage Verfahren betrifft Abhängigkeiten zwischen Attributen. Das Verfahren geht von der Annahme aus, dass zwischen den Attributen keine Abhängigkeiten bestehen. Neben offensichtlichen Abhängigkeiten zwischen Attributen wie z.B. bei Postleitzahl und Ort, existieren auch nicht unmittelbar erkennbare. Bei Vor- und Nachnamen können z.B. Kombinationen vorkommen, die besonders oft auftreten. So kommt z.B. der Vorname „Marcel“ häufiger mit dem Nachnamen „Duchamp“ als mit „Müller“ vor. Ein Ansatz, dieses Problem zu umgehen, besteht in der Kombination abhängiger Attribute zu einem gemeinsamen.[405]

Regelbasierte Erkennungsverfahren benutzen domänenspezifische Regeln, um bei einem Vergleich zwischen zwei Datensätzen Duplikate zu erkennen. Sie haben zwar den Vorteil bei der Erzeugung effizienter Regeln eine hohe Erkennungsrate (Precision) zu haben, allerdings ist hierfür ein extrem hoher manueller Aufwand notwendig.[406]

Bei dem regelbasierten Ansatz von IntelliClean, einem an der Universität Singapur entwickelten System, werden nicht nur Ähnlichkeiten zwischen Attributen zur Erkennung

[405] Gill, L.: Methods for Automatic Record Matching and Linkage and their Use in National Statistics, National Statistics Methodological, Series No. 25; National Statistics; London; 2001; S. 63f

[406] Elmagarmid, A.K.; Ipeirotis, P.G.; Verykios, V.S.: Duplicate Record Detection: A Survey; erschienen in: IEEE Transactions on Knowledge and Data Engineering, Vol. 19, No. 1; IEEE Computer Society; Washington, DC; 2007; S. 10

herangezogen, sondern Regeln zur Überprüfung der Ähnlichkeit zwischen zwei Datensätzen domänenspezifisch erstellt.[407] Eine Regel bei IntelliClean sieht beispielhaft wie folgt aus:[408]

```
DEFINE RULE COMPANY_RULE
        INPUT RECORDS : A, B
        IF  (A.currency ==B.currency)                AND
            (A.telephone == B.telephone)             AND
            (A.telephone != EMPTY_STRING)            AND
            (SUBSTRING_ANY(A.code, B.code) == TRUE) AND
            (FIELDSIMILARITY(A.address, B.address) > 0.85)
        THEN
                DUPLICATES(A, B), CERTAINTY = 0,85
```

Ein weiterer regelbasierter Ansatz, als "Equational Theory" bezeichnet, findet sich bei der von M.A. Hernandez und S.J. Stolfo entwickelten Sorted-Neigborhood Methode (SNM), auf die bereits in Abschnitt 3.5.3.2 über die Reduktion des Suchraumes eingegangen wurde. Bei der Erkennung von Duplikaten werden bei der SNM deklarativ Regeln in einer natürlichen Regelsprache aufgestellt. Eine Regel bei der SNM sieht beispielsweise wie folgt aus:

```
IF the last name of r1 equals the last name of r2,
    AND the first names differ slightly,
    AND the address of r1 equals the address of r2
THEN
        r1 is equivalent to r2
```

Die Implementation von "differ slightly" erfolgt dabei über eine Distanzfunktion und einen vorher definierten Schwellwert.[409]

Neben den bisher beschriebenen Verfahren wurden auch Methoden des maschinellen Lernens verwendet, um Duplikate zu erkennen.[410] Dazu werden in der Regel Duplikate einer Stich-

[407] Vgl. hierzu: Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 122

[408] Low, W.L.; Lee, M.L.; Ling, T.W.: A Knowledge-Based Approach for Duplicate Elimination in Data Cleaning; erschienen in: Information Systems, Vol. 26, Issue 8 (December 2001); Elsevier Science; Oxford, UK; S. 594

[409] Hernandez, M.A.; Stolfo, S.J.: Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem; Data Mining and Knowledge Discovery, Vol. 2, No. 1; Kluwer Academic; Boston; 1998; S. 16

probe manuell klassifiziert, um daraufhin einen Klassifizierer auf dieser Stichprobe lernen zu lassen. Daraufhin können über diesen Klassifizierer Duplikate in dem gesamten Datenbestand erkannt werden.[411]

L. Gravano u.a. beschreiben zwei Verfahren, um einen approximativen Join bei Datenbanken durchzuführen. Das erste verwendet hierzu in einem zweistufigen Algorithmus N-Grams, die in Datenbanktabellen gespeichert werden und Edit Distanzen.[412] Das zweite Verfahren nutzt den TF-IDF Algorithmus zur Gewichtung einzelner Token, die dann mit ihrem Gewicht in Tabellen der Datenbank gespeichert werden.[413]

In Abschnitt 3.5.3.2 zur Reduktion des Suchraumes wurde beschrieben, wie die Duplikaterkennung weniger Datensatzvergleiche durchführen muss, um effizienter zu sein. Entsprechende Verbesserungen sollten auch beim Datensatzvergleich berücksichtigt werden. Ist z.B. bei einem Datensatzpaarvergleich bereits der Schwellwert zur Erkennung eines Duplikats erreicht, müssen die übrigen Attribute nicht mehr verglichen werden. Andere Möglichkeiten bestehen in der Einschränkung der zu vergleichenden Attribute z.B. durch Verfahren zur Feature Selection.[414]

---

[410] Vgl. hierzu: Elmagarmid, A.K.; Ipeirotis, P.G.; Verykios, V.S.: Duplicate Record Detection: A Survey; erschienen in: IEEE Transactions on Knowledge and Data Engineering, Vol. 19, No. 1; IEEE Computer Society; Washington, DC; 2007; S. 7f

[411] Vgl. hierzu: Verykios, V.; Elmagarmid, A.: Automating the Approximate Record Matching Process, Purdue University, West Lafayette, 1999
Elfeky, M. G.; Verykios, V.; Elmagarmid, A K..: TAILOR: A Record Linkage Toolbox; erschienen in: Proceedings of the International Conference on Data Engineering (ICDE); San Jose; 2002
Borthwick, A.: Probabilistic Record Linkage Model derived from Training Data, United States Patent Application Publication, o.A., 2003

[412] Vgl. hierzu: Gravano, L.; Ipeirotis, P.G.; Jagadish, H.V.; Koudas, N.; Muthukrishnan, S.; Srivastava, D.: Approximate String Joins in a Database (Almost) for Free; erschienen in: Proceedings of the 27th Very Large DataBase (VLDB) Conference; Rom, Italien; 2001; S. 1

[413] Vgl. hierzu: Gravano, L.; Ipeirotis, P.G.; Koudas, N.; Srivastava, D.: Text Joins for Data Cleansing and Integration in an RDBMS; erschienen in: Proceedings of the 19th IEEE International Conference on Data Engineering (ICDE 2003); Bangalore, Indien; 2003

[414] Vgl. hierzu: Elmagarmid, A.K.; Ipeirotis, P.G.; Verykios, V.S.: Duplicate Record Detection: A Survey; erschienen in: IEEE Transactions on Knowledge and Data Engineering, Vol. 19, No. 1; IEEE Computer Society; Washington, DC; 2007; S. 12

3.5.3.4 Duplikate zusammenführen

Wurden Duplikate erkannt, müssen die Gruppen von Duplikaten zu jeweils einem Datensatz zusammengeführt und Konflikte zwischen Werten der Datensätze aufgelöst werden.[415] Dieser Schritt wird häufig auch als Datenfusion oder Record Merging bezeichnet. Im Rahmen dieser Arbeit wird in Anlehnung an die Duplikaterkennung dieser Schritt als Duplikatfusion bezeichnet.

Im einfachsten Fall können aus einer Datensatzgruppe von Duplikaten bis auf einen Datensatz alle gelöscht werden. Sinnvoller jedoch ist die Kombination von Datenwerten, um einen vollständigeren fusionierten Datensatz zu erhalten.[416] Bei der Zusammenführung von Duplikaten werden also nicht nur Duplikate entfernt, sondern Datensätze eventuell auch angereichert, sofern sich die Daten der zusammenzuführenden Datensätze gegenseitig ergänzen.[417] Ziel der Zusammenführung sollte es sein, einen Informationsverlust möglichst zu vermeiden. Aus Tabelle 21 können die Kombinationsmöglichkeiten bei der Fusion von zwei Attributen entnommen werden.

Tab. 21: Kombinationsmöglichkeiten bei der Duplikatfusion

	Wert Datensatz 1	Wert Datensatz 2	Fusionierter Wert
(a)	NULL	NULL	NULL
(b)	X	NULL	X
(c)	X	X	X
(d)	X	Y	Auflösungsfunktion

Quelle: eigene Darstellung

Haben alle zusammengehörigen Duplikate für ein Attribut keinen Wert gespeichert, weil dieser unbekannt ist (also NULL), wird im fusionierten Datensatz auch NULL übernommen (a). Enthält genau ein Datensatz für ein Attribut einen Wert, so wird dieser Wert im fusionierten Datensatz verwendet (b). Ist bei mehreren Datensätzen für ein Attribut der

[415] Naumann, F.; Häussler, M.: Declarative Data Merging with Conflict Resolution; erschienen in: Proceedings of the 7th International Conference on Information Quality (ICIQ-02); Cambridge, Massachusetts, USA; 2002; S. 2
[416] Naumann, F.: Datenqualität; erschienen in: Informatik-Spektrum; Vol. 30, No. 1, Februar 2007; Springer Verlag; Heidelberg, Berlin; 2007, S. 30
[417] Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 343f

gleiche Wert gespeichert, so wird dieser übernommen (c). Enthält ein Attribut dagegen bei mehreren Datensätzen unterschiedliche Werte, so liegt ein Datenkonflikt vor und eine Auflösungsfunktion muss den fusionierten Wert bestimmen (d).[418]

Datenkonflikte innerhalb einer Tabelle entstehen immer dann, wenn bei der Datenerfassung nicht oder ungenügend geprüft wurde, ob bereits ein entsprechender Datensatz in der Tabelle existiert.[419] Eine Auflösungsfunktion muss diese Datenkonflikte lösen, indem es aus zwei oder mehreren Werten einen neuen Wert ermittelt. Dazu kann sie als Eingabe auch andere Werte dieser Domäne oder anderer Attribute, die in Zusammenhang mit dem zu untersuchenden Attribut stehen, verwenden. Beispielsweise kann bei einem Attribut Preis, das unterschiedliche Werte bei den einzelnen Duplikaten enthält, ein vorhandenes Attribut Datum mit berücksichtigt werden, um z.B. den aktuellsten Preis zu verwenden.[420] Einfache Auflösungsfunktionen für numerische Attribute können z.B. den Mittelwert, das Maximum oder Minimum als Auflösungswert verwenden. Bei Zeichenketten können z.B. alle Zeichenketten der Duplikate im fusionierten Datensatz als ein Wert gespeichert werden.

Die folgende Tabelle 22 führt mehrere Auflösungsfunktionen auf.

[418] Naumann, F.; Häussler, M.: Declarative Data Merging with Conflict Resolution; erschienen in: Proceedings of the 7th International Conference on Information Quality (ICIQ-02); Cambridge, Massachusetts, USA; 2002; S. 3
[419] Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 345
[420] Naumann, F.; Häussler, M.: Declarative Data Merging with Conflict Resolution; erschienen in: Proceedings of the 7th International Conference on Information Quality (ICIQ-02); Cambridge, Massachusetts, USA; 2002; S. 3

Tab. 22: Auflösungsfunktionen bei der Duplikatfusion

Datentyp	**Funktion**	**Beschreibung**
Numerisch	SUM	Summe aller Eingabewerte, z.B. Einkommen
Numerisch	MEDIAN	Median
Numerisch	AVG	Arithmetischer Mittelwert, z.B. bei Bewertungen
Numerisch	VAR	Varianz
Numerisch	STDDEV	Standardabweichung
Text	MINLENGTH	Kürzester Text
Text	MAXLENGTH	Längster Text
Text	CONCAT	Verkettung aller Eingabewerte zu einem Text
Text	ANNCONCAT	Kommentierte Verkettung aller Eingabewerte zu einem Text, indem z.B. der ursprüngliche Primärschlüssel vor dem eigentlichen Text mit gespeichert wird
Text/Numerisch	COUNT	Anzahl von NOT NULL-Werten, die aktuellen Werte gehen verloren und die ursprünglichen Werte werden in einem zweckgebundenen Attribut gespeichert
Text/Numerisch	MAXFREQ	Wert, der am häufigsten vorkommt (nur bei einer hohen Anzahl von Duplikaten sinnvoll)
Text/Numerisch	MAX	Maximaler Wert
Text/Numerisch	MIN	Minimaler Wert
Text/Numerisch	RANDOM/ANY	Zufallsbasierter Wert aus den Eingabewerten
Text/Numerisch	MOSTRECENT	Zeitlich jüngster Wert
Text/Numerisch	FIRST/COALESCE	Übernimmt den ersten Wert, der von NULL verschieden ist
Text/Numerisch	LAST	Übernimmt den letzten Wert, der von NULL verschieden ist
Text/Numerisch	DISCARD	Übernimmt den Wert nur, wenn er für alle übereinstimmt, ansonsten NULL

Quelle: eigene Darstellung in Anlehnung an: Naumann, F.; Häussler, M.: Declarative Data Merging with Conflict Resolution; erschienen in: Proceedings of the 7th International Conference on Information Quality (ICIQ-02); Cambridge, Massachusetts, USA; 2002; S. 5
Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 353
Müller, H.; Weis, M.; Bleiholder, J.; Leser, U.: Erkennen und Bereinigen von Datenfehlern in naturwissenschaftlichen Daten; erschienen in: Datenbank-Spektrum, 5. Jahrgang, Heft 15; dpunkt Verlag; Heidelberg; 2005; S. 34

In relationalen Datenbankmanagementsystemen erfolgt eine Fusion von Datensätzen in SQL am einfachsten über den UNION-Operator. Dabei werden jedoch nur Datensätze zusammengeführt, die identische Werte haben.[421] Einen SQL-Operator, der die oben beschriebenen Zusammenführungen (a) bis (d) durchführt, gibt es im SQL-Standard nicht. Gruppierungen bzw. Zusammenführungen sind zwar mit dem GROUP BY-Operator und Aggregationsfunktionen möglich, allerdings durch die geringe Anzahl an Aggregationsfunktionen nur sehr beschränkt.[422] Hier ist der Einsatz von benutzerdefinierten Aggregationsfunktionen sinnvoll, die zwar von kommerziellen Datenbankmanagementsystemen angeboten werden, aber noch nicht in SQL:2003 standardisiert sind. Eine Möglichkeit ein RDBMS um eigene Aggregatfunktionen mit neuen an SQL angelehnten Sprachkonstrukten zu erweitern, wird von H. Wang und C. Zaniolo beschrieben, die eine eigene Sprache, die Aggregate Extension Language (AXL), hierfür spezifizieren[423].

J. Bleiholder und F. Naumann beschreiben zur Datenfusion und zur Auflösung von Datenkonflikten einen eigenen SQL-Operator, über den eine virtuelle Datenintegration möglich ist. Über den an den GROUP BY Operator angelehnten FUSE BY werden mehrere Datensätze zu einem Datensatz zusammengefasst und Datenkonflikte gelöst[424]. Ein einfaches Beispiel für den FUSE BY Operator sieht wie folgt aus:

```
SELECT Name, RESOLVE(Vorname), RESOLVE(Alter, MAX)
FROM Kunde, Adressen
FUSE BY Name ON ORDER Kunde.Alter DESC
```

[421] Naumann, F.: Informationsintegration - Antrittsvorlesung, Heft 134; Humboldt-Universität zu Berlin; Berlin; 2004; S. 17

[422] Naumann, F.: Informationsintegration - Antrittsvorlesung, Heft 134; Humboldt-Universität zu Berlin; Berlin; 2004; S. 18

[423] vgl. hierzu: Wang, H.; Zaniolo, C.: Using SQL to Build New Aggregates and Extenders for Object-Relational Systems; erschienen in: Abbadi, A. E.; Brodie, M. L.;Chakravarthy, S.; Dayal, U.; Kamel, N.; Schlageter, G.; Whang, K.-Y. (Hrsg.): Proceedings of 26th International Conference on Very Large Data Bases (VLDB); September 10-14, 2000; Cairo, Ägypten und
Wang, H.; Zaniolo, C.: User Defined Aggregates in Object-Relational Systems; erschienen in: 16th International Conference on Data Engineering (ICDE'2000); IEEE Computer Society 2000; San Diego, USA; 2000; S. 135-144

[424] vgl. hierzu: Bleiholder, J.;Naumann, F.: Declarative Data Fusion - Syntax, Semantics and Implementation; erschienen in: Eder, J.; Haav, H.-M.; Kalja, A.; Penjam, J. (Hrsg.): Advances in Databases and Information Systems 9th East European Conference, Tallinn, Estonia; September 12-15, 2005; Proceedings; Lecture Notes in Computer Science (LNCS), Vol. 3631, S. 58-73 und
Bleiholder, J.: A Relational Operator Approach to Data Fusion, 31st International Conference on Very Large Data Bases (VLDB) 2005 PhD Workshop; Trondheim; Norway; September 2005

Über das das Schlüsselwort RESOLVE werden Spalten mit eventuell vorhandenen Konflikten und Möglichkeiten ihrer Auflösung angegeben. Wird keine Auflösungsfunktion angegeben, folgt in Anlehnung an die SQL-Funktion COALESCE eine Rücklieferung des ersten NOT NULL-Wertes. Über das ON ORDER Schlüsselwort wird die Reihenfolge bestimmt, in der Werte von einer Konfliktauflösungsfunktion bearbeitet werden. Im Gegensatz zum GROUP BY unterstützt der FUSE BY Operator mehr als eine Tabelle und variable Auflösungsfunktionen bei Datenkonflikten anstelle von Aggregatfunktionen[425]. Der FUSE BY Operator setzt bereits voraus, dass eine Duplikaterkennung erfolgt und eine gemeinsame Spalte zur Identifikation gleicher Datensätze vorhanden ist (im Beispiel die Spalte „Name“).[426] Als FUSE BY-Klausel wird der Objektidentifizierer verwendet, der angibt welche Datensätze auf das gleiche reale Objekt verweisen.[427] Eine Implementierung des FUSE BY Operators findet sich in dem System HumMer (Humboldt-Merger), das innerhalb des DFG-Forschungsprojektes „MAC: Merging Autonomous Content and the HumMer (Humboldt-Merger)“ am Hasso-Plattner-Institut entwickelt wird.

L.L. Yan und M.T. Özsu beschreiben eine Erweiterung von SQL in der Form konflikttoleranter Abfragen („conflict tolerant“ - CT). Dabei werden drei Ebenen von Konflikttoleranzen unterschieden: HighConfidence, RandomEvidence und PossibleAtAll. Zur Konfliktauflösung werden Aggregat- oder benutzerdefinierte Funktionen verwendet. Über die Konflikttoleranzen wird in der WHERE-Klausel festgelegt, welche Datensätze das Prädikat der Gruppe der zusammenzuführenden Datensätze erfüllen müssen. Bei HighConfidence müssen alle Datensätze der Datensatzgruppe das Prädikat erfüllen, bei PossibleAtAll mindestens ein Datensatz und bei RandomEvidence wird ein beliebiger Datensatz der Datensatzgruppe gewählt. Falls dieser das Prädikat erfüllt, wird das Prädikat als wahr ausgewertet. Dazu ein Beispiel:

```
SELECT PID, Name[ANY], Adresse[DISCARD], Alter[AVG]
FROM Kunden
WHERE Alter > 30 WITH HighConfidence
```

[425] Bleiholder, J.; Naumann, F.:FUSE BY: Syntax und Semantik zur Informationsfusion in SQL. Informatik 2004 Workshop über Dynamische Informationsfusion.; Ulm, Germany; September 2004, S. 7ff

[426] vgl. hierzu: Müller, H.; Weis, M.; Bleiholder, J.; Leser, U.: Erkennen und Bereinigen von Datenfehlern in naturwissenschaftlichen Daten; erschienen in: Datenbank-Spektrum, 5. Jahrgang, Heft 15; dpunkt Verlag; Heidelberg; 2005; S. 33

[427] Bilke, A.; Bleiholder, J.; Böhm, C.; Draba, K.; Naumann, F.; Weis, M.: Automatic Data Fusion with HumMer; erschienen in: Proceedings of the 31st Very Large Data Bases (VLDB) Conference; Trondheim, Norwegen; 2005; S. 2

Alle Datensätze der Datensatzgruppe müssen das Prädikat (Alter > 30) erfüllen, damit das Prädikat wahr ergibt (HighConfidence). Bei den ausgewählten Datensätzen wird bei Konflikten entweder ein beliebiger Wert (ANY) gewählt oder NULL, falls nicht alle Werte eindeutig sind (DISCARD) bzw. beim Alter der arithmetische Mittelwert gebildet. Die beschriebene Datenfusion und Auflösung von Datenkonflikten wurde im AURORA System implementiert, einem Mediator Framework zur Integration von Datenquellen.[428]

3.5.3.5 Metriken zur Überprüfung

Um die Qualität von Verfahren zur Duplikaterkennung zu vergleichen, benötigt man Kennzahlen zu deren Bewertung. Geht man bei der Duplikaterkennung von einer Gruppe der Matches (Duplikate = Datensatzpaare, die wirklich zusammengehören) und einer Gruppe Unmatches (Nicht-Duplikate = Datensatzpaare, die nicht zusammengehören), kann man zwei Arten von Fehlern unterscheiden:

Duplikate, die keine sind, also fälschlicherweise als solche erkannt wurden (false-positive) und Datensatzpaare, die fälschlicherweise als Nicht-Duplikate erkannt wurden (false-negative).[429] Folgende Zuordnungen eines Datensatzes sind also möglich:

Tab. 23: Klassifikationsmatrix der Duplikaterkennung

Realität / Klassifikation	Match	Unmatch
Match	true-positives (TP)	false-positives (FP)
Ummatch	false-negatives (FN)	true-negatives (TN)

Quelle: eigene Darstellung in Anlehnung an: Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 332

[428] Vgl. hierzu: Yan, L.L.; Özsu, M. T.: Conflict Tolerant Queries in AURORA; erschienen in: Proceedings of the Fourth IECIS International Conference on Cooperative Information Systems; IEEE Computer Society; Washington, DC, USA; 1999; S. 279-290

[429] Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 331ff

Bezeichnet man die korrekt erkannten Duplikate als *TP* und die korrekt als Nicht-Duplikate erkannten Datensätze als *TN*, so setzen sich die „wahren“ Matches *M* und Unmatches *U* wie folgt zusammen:

$$M = TP + FN$$

$$U = FP + TN$$

wobei

M Match

U Unmatch

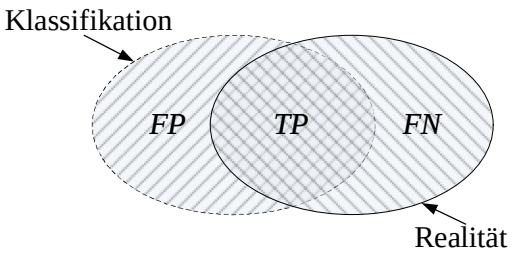
Um die Effektivität einer Duplikaterkennung zu überprüfen, gibt es aus dem Forschungsbereich des Information Retrieval zwei bekannte Metriken: *Recall* und *Precision*. *Recall* misst, wie viele wirkliche Matches (*TP*) im Verhältnis zu allen Matches erkannt wurden, also wie hoch der Anteil der korrekt erkannten Duplikate ist:

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{M}$$

Precision misst, wie viele wirkliche Matches (*TP*) im Verhältnis zu allen mit dem Verfahren erkannten Matches identifiziert wurden:

$$Precision = \frac{TP}{TP + FP}$$

Abb. 28: Recall und Precision



$$\text{Recall} = \frac{TP}{TP + FN} \qquad \text{Precision} = \frac{TP}{TP + FP}$$

Quelle: eigene Darstellung

Recall und *Precision* können wie folgt als harmonisches Mittel zu der Kennzahl *F-Score* kombiniert werden, um eine zusammenfassende Bewertungskennzahl zu erhalten:[430]

$$F\text{-}Score = 2 \cdot Recall \cdot \frac{Precision}{Precision + Recall}$$

Der *F-Score* ist 0, wenn keine wahren Dubletten erkannt wurden und 1 wenn alle erkannten Duplikaze auch wahre Duplikate sind. *F-Score* ergibt nur dann einen hohen Wert, wenn sowohl *Recall* als auch *Precision* hoch sind und stellt damit sozusagen einen Kompromisswert zwischen diesen beiden Metriken dar.[431]

Die von S. Melnik, H. Garcia-Molina und E. Rahm vorgeschlagene *Accuracy*-Metrik kombiniert *Recall* und *Precision* mit der Einschränkung, dass *Precision* nicht kleiner als 0,5 sein sollte. D.h. die Hälfte der erkannten Matches sollte korrekt sein, ansonsten ist *Accuracy* negativ.[432] *Accuracy*, auch als Overall bezeichnet, berücksichtigt damit den Aufwand, falsch erkannte Duplikate zu entfernen und fehlende hinzuzufügen.[433]

$$Accuracy = Recall \cdot \left(2 - \frac{1}{Precision}\right)$$

J.C. van Rijsbergen kombiniert *Recall* und *Precision*, indem er ein Variable *b* einführt, die die relative Wichtigkeit von *Precision* und *Recall* durch den Anwender festlegt:

$$E = 1 - \frac{(1 + b^2) \cdot Precision \cdot Recall}{b^2 \cdot Precision + Recall}$$

[430] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 126f
[431] Baeza-Yates, R.; Ribeiro-Neto, B.: Modern Information Retrieval; Addison-Wesley; Harlow; 1999; S. 82
[432] Melnik, S.; Garcia-Molina, H.; Rahm, E.: Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching; erschienen in: Proceedings of the 18th ICDE; San Jose; 2002
[433] Do, H.H.; Rahm, E.: COMA A system for flexible combination of Schema Matching Approaches; erschienen in: Proceedings of the 28.th Very Large Data Bases (VLDB) Conference; Hong Kong; 2002; S. 8

Hat *b* den Wert 0,5, ist der Anwender 2mal mehr an *Precision* als an *Recall* interessiert, bei einem Wert von 2 dagegen 2mal mehr an *Recall* als an *Precision*.[434]

Daneben gibt es noch das weniger bekannte Maß des *Fallout* („Abfallquote"), das Verhältnis von fälschlicherweise erkannten Duplikaten zu allen Datensätzen der Gruppe Unmatch:[435]

$$Fallout = \frac{FP}{FP + TN} = \frac{FP}{U}$$

Im folgenden Beispiel repräsentieren 10 Datensätze sechs unterschiedliche Datensätze der realen Welt:

A1, A2, A3, B1, B2, B3, C1, D1, E1, F1

wobei

{A1, A2, A3} und {B1, B2, B3} wahre Duplikate sind

Bei einer Duplikaterkennung wurden {A1, A2, C1} und {B1, B2} als Duplikate erkannt. Vier Datensätze davon wurden als wahre Duplikate und C1 und B3 wurden nicht korrekt erkannt.[436] Damit ergibt sich folgende Klassifikationsmatrix mit den entsprechenden Metriken:

[434] Frakes, W.B.: Introduction to Information Storage and Retrieval Systems; erschienen in: Frakes, W.B.; Baeza-Yates, R. (Hrsg.): Information Retrieval - Data Structures & Algorithms; Prentice-Hall; Upper Saddle River, New Jersey; 1992; S. 11

[435] Manning, C. D., Schütze, H.: Foundations of Statistical Natural Language Processing; The MIT Press; Cambridge, Massachusetts; 2003; S. 270

[436] Vgl. hierzu: Low, W.L.; Lee, M.L.; Ling, T.W.: A Knowledge-Based Approach for Duplicate Elimination in Data Cleaning; erschienen in: Information Systems, Vol. 26, Issue 8 (December 2001); Elsevier Science; Oxford, UK; S. 587

Tab. 24: Beispiel zur Bewertung eines Duplikaterkennungsverfahrens

Realität / Erkennung	Match	Unmatch
Match	A1, A2, B1, B2	C1
Unmatch	A3, B3	D1, E1, F1

$$\textit{Recall} = \frac{4}{4+2} = 66{,}7\% \qquad \textit{Precision} = \frac{4}{4+1} = 80{,}0\%$$

$$\textit{F-Score} = 2 \cdot 66{,}7\% \cdot \frac{80{,}0\%}{80{,}0\% + 66{,}7\%} = 72{,}72\%$$

$$\textit{Accuracy} = 66{,}7\% \cdot \left(2 - \frac{1}{80{,}0\%}\right) = 50{,}0\%$$

$$\textit{Fallout} = \frac{1}{1+3} = 25{,}0\%$$

$$E = 1 - \frac{(1+0{,}5^2) \cdot 80\% \cdot 66{,}7\%}{0{,}5^2 \cdot 80\% + 66{,}7\%} = 76{,}92\% \quad \text{mit } b = 0{,}5$$

$$E = 1 - \frac{(1+2^2) \cdot 80\% \cdot 66{,}7\%}{2^2 \cdot 80\% + 66{,}7\%} = 68{,}97\% \quad \text{mit } b = 2$$

Quelle: eigene Darstellung

Je mehr Datensatzpaare mit einem geringen Ähnlichkeitsgrad als Duplikate akzeptiert werden, umso höher ist der *Recall*-Wert, allerdings auf Kosten des *Precision*-Wertes. Umgekehrt ist der *Precision*-Wert umso höher, je größer der Ähnlichkeitsgrad zwischen zwei Datensätzen sein soll. Dieses Problem wird als das *Recall-Precision*-Dilemma bezeichnet.[437] Generell kann man sagen, dass tolerante Erkennungsmethoden zu einem hohen *Recall*, aber niedriger *Precision* führen und umgekehrt strenge Erkennungsmethoden zu einer hohen *Precision*, aber einem niedrigen *Recall*.[438]

[437] Low, W.L.; Lee, M.L.; Ling, T.W.: A Knowledge-Based Approach for Duplicate Elimination in Data Cleaning; erschienen in: Information Systems, Vol. 26, Issue 8 (December 2001); Elsevier Science; Oxford, UK; S. 585

[438] Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 333

Die hier vorgestellten Metriken werden nicht nur zur Beurteilung von Verfahren zur Duplikaterkennung verwendet, sondern sind vor allem auch zur Beurteilung der Vorhersage von maschinellen Lernverfahren oder bei der Suche im Information Retrieval als Qualitätsindikator geeignet.

3.5.4 Validierung

Die Validierung der Daten erfolgt durch die Überprüfung der Einhaltung von Regeln. Regeln werden dabei nicht nur geprüft (z.B. Bereichsüberprüfung), sondern es können die Daten auch korrigiert oder Ausreißer durch ihre „wahren" Werte, logische Werte oder NULL ersetzt werden. Eine Regel, die automatisch korrigiert, kann z.B. bei numerischen Werten NULL-Werte durch den Mittelwert ersetzen oder einen Klassifizierer zur Bestimmung eines Wertes verwenden. Eine Ersetzung von NULL-Werten durch den am häufigsten auftretenden Wert kann daneben auch sinnvoll sein. Nach C. Batini und M. Scannapieco werden diese Verfahren als „Error Correction"[439] oder Imputation[440] bezeichnet. Semantische Regeln werden auf einen Datenbestand angewendet und die Daten in Datensätzen, die den Regeln nicht entsprechen mit Inhalten korrigiert, die zur Regeleinhaltung führen. Daten werden also auf Grund von Regeln angepasst und aktualisiert.

Beispiel: IF Alter=6 AND Familienstand=verheiratet THEN Familienstand=ledig

Zu beachten ist, dass das Ersetzen von Daten durch Algorithmen durchaus die Semantik der Daten verfälscht. Wird z.B. in einer Regel das Geschlecht durch das im Datenbestand am häufigsten auftretende Geschlecht bei NULL-Werten ersetzt, so kann dies natürlich für den Datenbestand inkorrekt sein, da der Wert geschätzt wurde[441].

[439] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 70
[440] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 83
[441] Vgl. hierzu: Dasu, T.; Johnson, T.: Exploratory Data Mining and Data Cleaning; John Wiley & Sons; Hoboken, New Jersey; 2003; S. 142

3.6 Steuern

3.6.1 Messen von Datenqualitätsmerkmalen

Existieren Regeln, so kann die Datenqualität auf Basis der Verletzung oder Einhaltung der Regeln gemessen werden. Werden Regeln verletzt, können diese zusätzlich mitprotokolliert werden, um Probleme zu identifizieren.[442] Die allgemeine Metrik einer Datenqualitätsmerkmals kann wie folgt aussehen:[443]

Bewertung = 1 – (Anzahl unerwünschter Ergebnisse / Anzahl aller Ergebnisse)

Zum Messen der Datenqualitätsmerkmale sollten Sollwerte definiert werden, die mit den Istwerten verglichen werden können. Auf hohen Aggregationsstufen schlagen M. Winter u.a. eine Ampelfunktion[444] wie folgt vor:

- die Daten sind verwendbar (grün)
- die Daten sind eingeschränkt verwendbar (gelb)
- die Daten sind nicht zu verwenden (rot)

Eine Möglichkeit, die Dimension Glaubwürdigkeit objektiv zu messen, schlägt F. Naumann vor. SELECT-Anweisungen sollen demnach automatisch analysiert werden, um zu erkennen, ob diese zweifelhafte Ergebnisse zurückliefern.[445] Das folgende einfache Beispiel einer syntaktisch korrekten SQL-Anweisung stellt z.B. eine SELECT-Anweisung dar, die niemals ein Ergebnis zurückliefert[446] und deren Anwendung damit als zweifelhaft angesehen werden muss:

---

[442] Winter, M.; Herrmann, C.; Helfert, M.: Datenqualitätsmanagement für Data-Warehouse-Systeme - Technische und organisatorische Realisierung am Beispiel der Credit Suisse; erschienen in: von Maur, E.; Winter, R. (Hrsg.): Data Warehouse Management: Das St. Galler Konzept zur ganzheitlichen Gestaltung der Informationslogistik; Springer; Heidelberg; 2003; S. 221-235

[443] Lee, Y.W.; Pipino, L.L.; Funk, J.D.; Wang, R.Y.: Journey to Data Quality; MIT Press; Cambridge, Massachusetts; 2006; S. 54

[444] Winter, M.; Helfert, M.; Herrmann, C.: Das metadatenbasierte Datenqualitätssystem der Credit Suisse; erschienen in: von Maur, E.; Winter, R. (Hrsg.): Vom Data Warehouse zum Corporate Knowledge Center - Proceedings der Data Warehousing 2002; Physica-Verlag; Heidelberg; 2002; S. 165

[445] Naumann, F.; Roth, M.: Information Quality: How good are Off-The-Shelf DBMS?; erschienen in: Proceedings of the 9th International Conference on Information Quality (ICIQ-04); Cambridge, Massachusetts, USA; S. 265

[446] Vgl. hierzu: Brass, S.; Goldberg, C.: Semantic Errors in SQL Queries: A Quite Complete List; erschienen in 4th International Conference on Quality of Software; Brunswick, Germany; 2004; S. 1

```
SELECT *
FROM Kunde
WHERE Name = 'Muster'  AND Name = 'Müller'
```

Voraussetzung zur Umsetzung eines solchen Ansatzes im RDBMS ist die Unterstützung eines ON SELECT im Trigger, das im SQL:2003 Standard aber nicht vorgesehen ist.

3.6.2 Aggregation von Datenqualitätsdimensionen

3.6.2.1 Datenqualitätspyramide

Metriken zur Datenqualität können in unterschiedlichen Einheiten und Wertebereichen gemessen sein. Um eine Aggregation von Datenqualitätsdimensionen vorzunehmen, müssen die Kennzahlen zunächst normiert werden (normalerweise in einen Wertebereich zwischen 0 und 1), indem z.B. durch den maximalen Sollwert dividiert wird. Eine Möglichkeit zur Normierung besteht in der linearen Skalierung, die nach folgender Formel umrechnet:[447]

$$v_n = \frac{v_i - \min(v_1,...,v_n)}{\max(v_1,...,v_n) - \min(v_1,...,v_n)} \cdot (newMax - newMin) + newMin)$$

mit

v_n	skalierter Wert
v_i	zu skalierender Wert
newMax	neuer Maximumwert
newMin	neuer Minimumwert

Sind Minimum und Maximum der Werte nicht hinreichend bekannt, ist es problematisch, da bei einem neuen Maximum oder Minimum die Bereichsbegrenzung zwischen 0 und 1 nicht mehr erfüllt ist. Bei der z-Transformation können die Werte deshalb auf Basis ihres Mittelwertes und ihrer Standardabweichung nach folgender Formel normiert werden:

[447] Han, J.; Kamber, M.: Data Mining - Concepts and Techniques; Morgan Kaufmann; San Francisco; 2001; S. 115

$$v_n = \frac{v_i - \overline{A}}{s_A}$$

mit

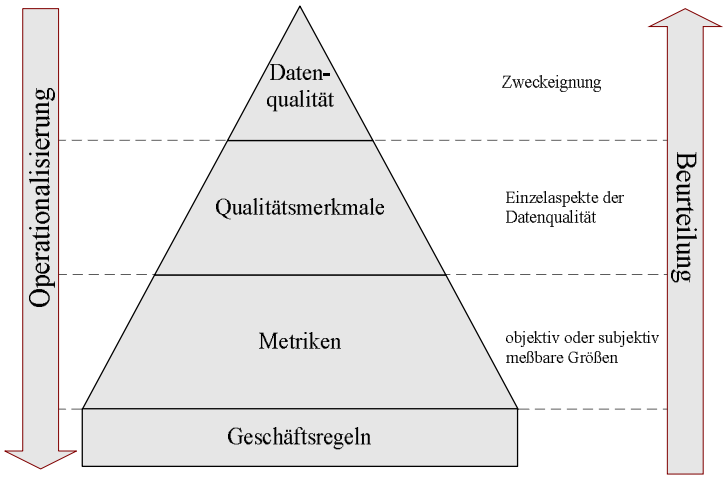
$\overline{A}$ Mittelwert des Attributs A

s_A Standardabweichung des Attributs A

Neben der Normierung ist auch eine Gewichtung der einzelnen Kennzahlen durch den Anwender sinnvoll.[448] Hierbei ist zu beachten, wie die Kennzahlen aggregiert werden können. Wahrscheinlichkeiten werden normalerweise multipliziert, bei Antwortzeiten ist es dagegen eher sinnvoll, einen Mittelwert oder den maximalen Wert zu verwenden. Preise wiederum sollten addiert werden.[449]

Um zu einer flexiblen Aussage über die Datenqualität zu kommen, ist es sinnvoll, Geschäftsregeln den vorher festgelegten Datenqualitätsmerkmalen zuzuordnen. Über diese Zuordnung kann dann eine Aussage über ein Merkmal erfolgen. Merkmale wiederum können aggregiert werden, um zu einer Gesamtaussage über mehrere Datenqualitätsmerkmale oder über die gesamte Datenqualität zu kommen.

Abb. 29: Aggregation zur Beurteilung der Datenqualität



Quelle: eigene Darstellung in Anlehnung an: Windheuser, U.: Datenqualität - Ein nicht unlösbares Problem; erschienen in: DMC Anwendertage 2004; Chemnitz; S. 3

[448] Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 361f
[449] Vgl. hierzu: Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 361

Zur Aggregation verschiedener Datenqualitätsdimensionen bieten sich Methoden zur multikriteriellen Entscheidungsfindung (Multiple Attribute Decision Making, MADM) an. K.P. Yoon und C.-L. Hwang beschreiben fünf Kriterien, für die MADM Methoden geeignet sind. Hierzu gehören:[450]

- Mehrere Alternativen
- Mehrere Attribute
- Unterschiedliche Einheiten
- Attributgewichte
- Abbildung in einer Entscheidungsmatrix

Die meisten der 17 von Yoon und Hwang beschriebenen MADM Methoden setzen einen Vergleich zwischen verschiedenen Alternativen voraus. Im Fall der Bewertung von Datenqualität bedeutet dies, dass sozusagen mehrere Datenquellen (Alternativen) bewertet werden müssen. Diese Eigenschaft gilt jedoch nicht für die Simple Additive Weighting (SAW) und die Weighted Product Methode. Diese beiden im Folgenden näher beschriebenen MADM Methoden haben also den Vorteil, dass zur Berechnung keine weiteren Alternativen notwendig sind.

J. Kübart u.a. beschreiben ein Verfahren, dass nicht die Datenqualitätsmerkmale zusammenfasst, sondern Regeln auf einzelne Datensätze aggregriert, um so zu einer Bewertung eines einzelnen Datensatzes als Ausreißer zu kommen. Dazu wird jeder Regel eine Priorität zugewiesen und dann überprüft, welcher Datensatz welche Regeln erfüllt. Bei Regeln, die nicht zutreffen, werden die Prioritäten zusammenaddiert, um als Indikator für eine ungenügende Datenqualität bewertet zu werden.[451]

3.6.2.2 Simple Additive Weighting (SAW)

Das einfachste Verfahren zur Aggregation mehrerer Kennzahlen ist die „Simple Additive Weighting“ (SAW) Methode, die aus drei Schritten besteht. Wie bereits im letzten Abschnitt

[450] Yoon, K.P.; Hwang, C.-L.: Multiple Attribute Decision Making; Sage Publications; Thousands Oaks, Kalifornien; 1995; S. 2f
[451] Kübel, J.; Grimmer, U.; Hipp, J.: Regelbasierte Ausreißersuche zur Datenqualitätsanalyse; erschienen in: Datenbank-Spektrum, 5. Jahrgang, Heft 14; dpunkt Verlag; Heidelberg; 2005; S. 24

beschrieben, müssen die einzelnen Dimensionskennzahlen zunächst normiert werden. Danach werden die einzelnen Kriterien gewichtet und im dritten Schritt die Summe der gewichteten Kriterien gebildet.[452] Die Berechnung sieht wie folgt aus[453]:

$$Q(p)=\sum_{i=1}^{n} q_i \cdot w_i$$

mit

$q_1,\ldots,q_n$ normierte Kennzahlen

$w_1,\ldots,w_i$ dimensionsspezifische Gewichtsfaktoren mit $\sum_{i=1}^{n} w_i = 1$

Dazu werden alle Kennzahlen mit ihren Gewichtsfaktoren multipliziert und die Summe daraus gebildet. Die Summe der Gewichte muss dabei den Wert 1 ergeben. Tabelle 25 zeigt noch einmal ein Beispiel zur Berechnung nach der SAW Methode.

Tab. 25: Beispiel zur SAW Methode

Kriterium	**Gewicht**	**Bewertung**	**Ergebnis**
Vollständigkeit	10%	3	0,3
Korrektheit	60%	8	4,8
Konsistenz	30%	6	1,8
Summe	**100%**		**8,4**

Quelle: eigene Darstellung

Bei der Berechnung ist zu berücksichtigen, dass die SAW Methode davon ausgeht, dass alle Attribute unabhängig voneinander sind.[454]

[452] Naumann, F.: Quality-Driven Query Answering for Integrated Information Systems, Dissertation; Springer Verlag; Berlin u.a.; 2002; S. 56
[453] Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007; S. 362
[454] Yoon, K.P.; Hwang, C.-L.: Multiple Attribute Decision Making; Sage Publications; Thousands Oaks, Kalifornien; 1995; S. 33

3.6.2.3 Weighted Product

Bei der SAW Methode müssen vor der Aggregation die Kriterien normalisiert werden. Bei der von Bridgam 1922[455] eingeführten Weighted Product Methode kann hierauf verzichtet werden, da die Kriterien multipliziert werden und die Gewichtung eines Kriteriums über einen zugeordneten Exponenten erfolgt.[456] Die Formel hierzu sieht wie folgt aus:

$$Q(p) = \prod_{i=1}^{n} q_i^{w_j}$$

mit

$q_1,\ldots,q_n$ Kennzahlen

$w_1,\ldots,w_i$ dimensionsspezifische Gewichtsfaktoren

Tabelle 26 führt eine Beispielrechnung zur Weighted Product Methode auf.

Tab. 26: Beispiel zur Weighted Product Methode

Kriterium	**Gewicht**	**Bewertung**	**Ergebnis**
Vollständigkeit	10%	3	1,12
Korrektheit	60%	8	3,48
Konsistenz	30%	6	1,71
Summe	**100%**		**6,31**

Quelle: eigene Darstellung

3.6.3 Monitoring

Die Analyse und die Verbesserung der Datenqualität ist ein kontinuierlicher Vorgang, d.h. Datenqualitätsmerkmale sollten überwacht und bei Abweichungen von einem vorgegebenen Sollwert zu einer Aktion führen. Neben dem chronologischen Messen der Datenqualitäts-

[455] Yoon, K.P.; Hwang, C.-L.: Multiple Attribute Decision Making; Sage Publications; Thousands Oaks, Kalifornien; 1995; S. 37

[456] Stallkamp, M.: Ziele und Entscheiden - Die Goalgetter-Methode und Goalgetter-Software, Diss.; Universität Duisburg-Essen, Wirtschaftswissenschaften; Essen; 2006; S. 73

kennzahlen und der Alarmierung bei einer Geschäftsregelverletzung gehört aber auch das Erkennen von Datentrends zum Monitoring.

Die gemessenen Datenqualitätskennzahlen können entweder als Metadaten mit den Daten selbst gespeichert werden oder als eigener Datenbestand. Zum Monitoring der gemessenen Kennzahlen ist es wichtig, dass bei Über- oder Unterschreiten einer Kennzahl vom vorgegebenen Schwellbereich, ein bestimmtes Ereignis eintreten muss, um das Problem zu analysieren oder falls möglich automatisch zu beheben. Im häufigsten Fall erfolgt eine Benachrichtigung, aber auch eine stufenweise Bearbeitung ist möglich. Eine Regel könnte z.B. festlegen, dass bei einer Abweichung von bis zu 10% bestimmte automatische Bereinigungsmaßnahmen durchgeführt werden sollen und erst darüber hinaus eine Benachrichtigung versendet wird.

4. Integration der Datenqualitätsverfahren

In Kapitel 2 wurde u.a. das Fünf-Schichten-Modell für Datenbankmanagementsysteme als Referenzarchitektur vorgestellt. Kernziele dieser Referenzarchitektur sind dabei vor allem die Datenunabhängigkeit zwischen konzeptioneller, logischer und physischer Schicht und Performanz des Systems. Im Speziellen wurden relationale Datenbankmanagementsysteme erläutert und dabei vor allem Aspekte aus Sicht der Anfragesprache näher beleuchtet, die zur Verbesserung der Datenqualität von Bedeutung sind. Hierzu gehören neben den semantischen Integritätsbedingungen auch prozedurale Sprachelemente und SQL-Routinen.

In Kapitel 3 wurden dann die Verfahren und Algorithmen beschrieben, die eine Datenqualitätskomponente enthalten sollte. Neben grundlegenden Verfahren wie Codealgorithmen, Proximitätsmaße, Tokenizer, Nachschlagetabellen, reguläre Ausdrücke, Segmentierung, Klassifizierung und vor allem Regeln gehören hierzu vor allem Verfahren zur allgemeinen Vorgehensweise, dem Verstehen der Daten, dem Verifizieren und Verbessern und dem Steuern der Datenqualität.

Im Folgenden geht es darum, eine generelle Entscheidung zu treffen, wie die in Kapitel 3 beschriebenen Verfahren in ein relationales Datenbankmanagementsystem integriert bzw. von diesem genutzt werden können. Dabei sollen drei verschiedene Architekturvarianten beschrieben und diskutiert werden:

- Einbettung in die bestehende Basisarchitektur
- Erweiterung der bestehenden Basisarchitektur
- Lose gekoppeltes Datenqualitätssystem

Unter der Einbettung einer Datenqualitätskomponente in die bestehende Basis- bzw. Referenzarchitektur wird hier die Einbindung dieser in die bestehenden Schichten der Referenzarchitektur verstanden. Hiervon betroffen sind vor allem die Schichten der Anfragesprache, der Anfrageverarbeitung, aber auch der Zugriffsplanverarbeitung, die zur Nutzung der Datenqualitätsverfahren angepasst werden müssen. Hierdurch ist auch eine Ergänzung der SQL-Syntax an die spezifischen Anforderungen von Datenqualitätsverfahren möglich.

H. Galhardas[457] u.a. beschreiben eine solche Erweiterung der SQL-Anfragesprache in ihrem Framework zur Datenintegration, genau wie Bleiholder[458] u.a. mit dem FUSE BY-Operator zur Datenfusion (vgl. hierzu Abschnitt 3.5.3.4).

Der Ansatz der Einbettung hat vor allem zwei wesentliche Vorteile: Zum einen wird die Benutzbarkeit durch SQL erleichtert, da es spezifische SQL-Sprachelemente zur Verbesserung der Datenqualität gibt. Zum anderen besteht aber auch, und das ist der wesentliche Vorteil, die Möglichkeit, Anfragen mit Datenqualitätsverfahren über den Anfrageoptimierer zu optimieren, um entsprechende Performanz zu erhalten. Problematisch ist dagegen die weitere Erhöhung des Sprachumfangs durch die Erweiterung von SQL. Es ist fraglich, ob diese Erweiterungen auch genutzt werden bzw. ihren Eingang in kommerzielle DBMS finden. Hinzu kommt, dass zur Nutzung der Erweiterungen alle bisherigen Anwendungssysteme, die auf das RDBMS aufsetzen, weitgehend angepasst werden müssten.

Der zweite Ansatz, die Erweiterung eines bestehenden relationalen Datenbankmanagementsystems, sieht vor, die bestehenden SQL-Erweiterungsmöglichkeiten eines RDBMS zu nutzen, um Datenqualitätsverfahren zu implementieren. Hierzu gehören vor allem die in Abschnitt 2.2.5 beschrieben Spracheigenschaften wie semantische Integritätsbedingungen, Trigger, SQL-Routinen (intern/extern) und Tabellenrückgabefunktionen. Dieses entspricht zwar auch einer mehr oder weniger umfangreichen Anpassung der SQL-Syntax, aber eben im Rahmen des aktuellen SQL-Standards. P.C. Lockemann und K.R. Dittrich bezeichnen diesen Ansatz als Aufbau, wenn die „neue Funktionalität auf der Grundlage existierender Funktionalität“ erweitert wird.[459]

Ein Vorteil dieses Ansatzes besteht in der Portabilität, da die Erweiterungsmöglichkeiten des SQL-Standards genutzt werden. Durch Verwendung von SQL-Routinen und Systemkatalogdaten ist es zudem möglich, Datenqualitätsanforderungen ohne Änderung der aufsetzenden Anwendungssysteme zu realisieren. Dadurch ist eine geradlinige Integration von Daten-

[457] Galhardas, H.; Florescu, D.; Shasa, D.; Simon, E.: An Extensible Framework for Data Cleaning (extended version); INRIA Technical Report; Sophia Antipolis; 1999 und
Galhardas, H.; Florescu, D.; Shasa, D.; Simon, E.; Saita, C.-A.: Declarative Data Cleaning: Language, Model and Algorithms; erschienen in: Proceeding of the 27th VLDB Conference; Roma, Italy; 2001
[458] Bleiholder, J.;Naumann, F.: Declarative Data Fusion - Syntax, Semantics and Implementation; erschienen in: Eder, J.; Haav, H.-M.; Kalja, A.; Penjam, J. (Hrsg.): Advances in Databases and Information Systems 9th East European Conference, Tallinn, Estonia; September 12-15, 2005; Proceedings; Lecture Notes in Computer Science (LNCS), Vol. 3631, S. 58-73
[459] Lockemann, P.C.; Dittrich, K.R.: Architektur von Datenbanksystemen; dpunkt Verlag; Heidelberg; 2004; S. 130

qualitätsverfahren in bestehende Anwendungssysteme mit geringfügigen oder gar keinen Änderungen an diesen möglich. Betrachtet man die Kernanforderungen der Referenzarchitektur, so besteht das Problem dieses Ansatzes darin, dass der Anfrageoptimierer nur bedingt Anfragen mit Datenqualitätsverfahren optimieren könnte.

Beim dritten Ansatz eines lose gekoppelten Datenqualitätssystems werden die Datenqualitätsverfahren in einem eigenständigen System implementiert (siehe Abb. 30). Die Verfahren werden über eine lokale Bibliothek oder über Netzwerkkommunikation wie z.B. RPC (Remote Procedure Call) oder SOAP aufgerufen. Dieser Ansatz entspricht von der Architektur dem Datenbasis-Verwalter für Metadaten, so wie ihn P.C. Lockemann und K.R. Dittrich beschreiben. Ein eigenständiger externer Metadatenverwalter kommuniziert mit den Komponenten des Datenbankmanagementsystems über definierte Schnittstellen.[460]

Der Vorteil dieses Dienstegeber-Ansatzes besteht darin, dass er für verschiedene Datenbankmanagementsysteme einsetzbar ist, allerdings mit dem gleichen Nachteil wie beim ersten Ansatz. Das Datenbankmanagementsystem muss zur Nutzung an verschiedenen Schichten und Komponenten geändert werden. Zur Nutzung ist daher ein hoher Änderungsaufwand notwendig. Hinzu kommt, dass ähnlich wie beim zweiten Ansatz ein Performanzverlust zu erwarten ist,[461] da zum einen eine Optimierung durch den Anfrageoptimierer kaum möglich ist und zudem eine Kommunikation mit einem externen System stattfinden muss.

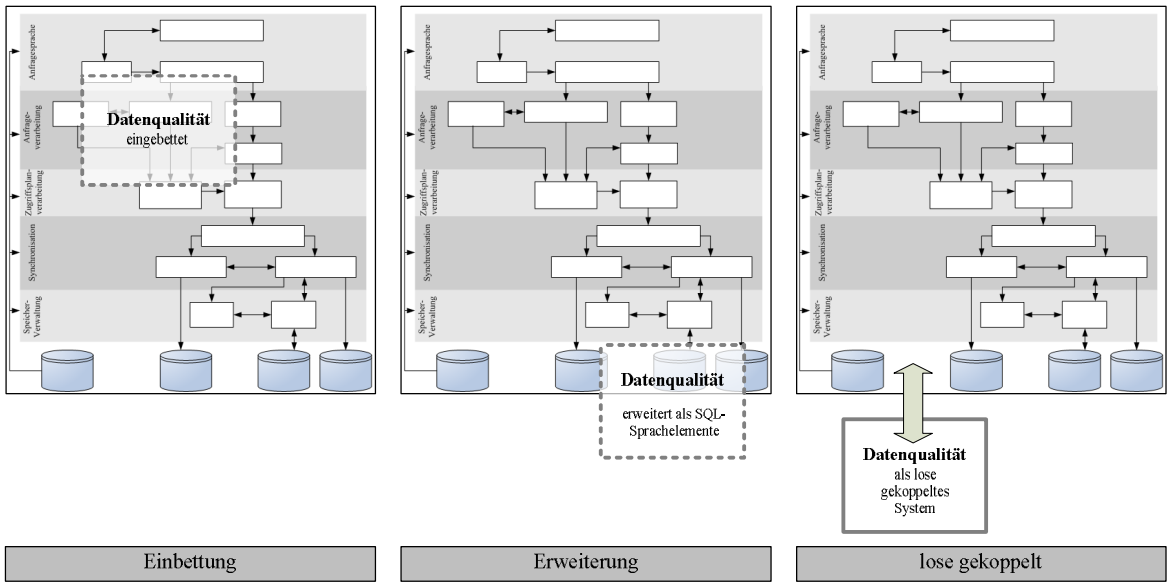
Betrachtet man Datenqualität wie in Kapitel 2 beschrieben als Eignung für einen Anwendungszweck, so kann man sie funktional eher als höheren Dienst betrachten, dessen Bedeutung in der Sicht des Anwenders liegt. Datenqualität ist letztendlich domänenspezifisch, insofern sollte sich diese Anforderung auch in der Architektur widerspiegeln. Die in Kapitel 2 beschriebene Kernanforderung der Performanz ist zwar generell wichtig, wird aber im Rahmen dieser Arbeit nur sekundär berücksichtigt, da es hier erst einmal um die Anforderung der Anwendbarkeit geht, also wie Datenqualitätsanforderungen funktional in ein RDBMS zu implementieren sind. Die Anforderung der Benutzbarkeit hat hier deshalb in der Architektur eindeutig Vorrang vor der Anforderung der Performanz. Die zweite wesentliche Anforderung besteht in der Erhaltung vorhandener Anwendungssysteme. Datenqualitätsverfahren sollen daher weitgehend auf RDBMS-Ebene deklariert werden können, ohne dass die auf das

[460] Lockemann, P.C.; Dittrich, K.R.: Architektur von Datenbanksystemen; dpunkt Verlag; Heidelberg; 2004; S. 61f

[461] Vgl. hierzu zur Performanz des Metadatenverwalters: Lockemann, P.C.; Dittrich, K.R.: Architektur von Datenbanksystemen; dpunkt Verlag; Heidelberg; 2004; S. 95f

RDBMS aufsetzenden Anwendungssysteme geändert bzw. nur geringfügig geändert werden müssen.

Abb. 30 Ansätze zur Integration von Datenqualitätsverfahren in ein RDBMS



Quelle: eigene Darstellung

Beide Anforderungen werden am ehesten von der ersten und zweiten Architekturvariante berücksichtigt, sofern man Änderungen an der SQL-Sprache erst einmal unberücksichtigt lässt. Der zweite Architekturansatz hat jedoch zusätzlich den Vorteil, einen Ansatz nach dem aktuellen SQL-Standard zu implementieren und ist insofern in seiner Umsetzung geradliniger, da er keine neuen Sprachelemente voraussetzt.

Im Folgenden wird der zweite Architekturansatz – Erweiterung der bestehenden Basisarchitektur – als Grundlage für die Erweiterung eines RDBMS um Datenqualitätsverfahren verwendet.

5. Architektur der Datenqualitätskomponente

5.1 Architektur

Wie bereits erwähnt, setzen Methoden zur Verbesserung der Datenqualität zum großen Teil Domänenwissen voraus. Primäre Ziele eines Datenqualitätsdienstes sollte deshalb Variabilität und Benutzbarkeit sein. Mit Variabilität ist die Möglichkeit gemeint, „Eigenschaften eines Systems mit geringem Aufwand zu ändern“. [462]

Softwarekomponenten, die die Softwarearchitektur vorgeben und anpassbar sind, bezeichnet man allgemein als Rahmenwerk (Framework). Ein Framework kann z.B. ein System kooperierender Klassen sein, über die das System anpassbar und erweiterbar wird. Zweck eines Frameworks ist vorrangig die Entwurfs-Wiederverwendung und nicht die Code-Wiederverwendung. Eine Möglichkeit besteht in der konkreten Implementierung von Unterklassen, um das Framework anzupassen.[463]

Ein Framework hat folgende drei Eigenschaften:[464]

- Umkehrung des Kontrollflusses
  (das Framework steuert den Kontrollfluss)
- Vorgabe einer konkreten Anwendungsarchitektur
  (Anpassen an die Anforderungen einer speziellen Anwendung)
- Anpassbarkeit durch Variationspunkte
  (Erweiterungen, unterschiedliche Konfigurationen u.ä.)

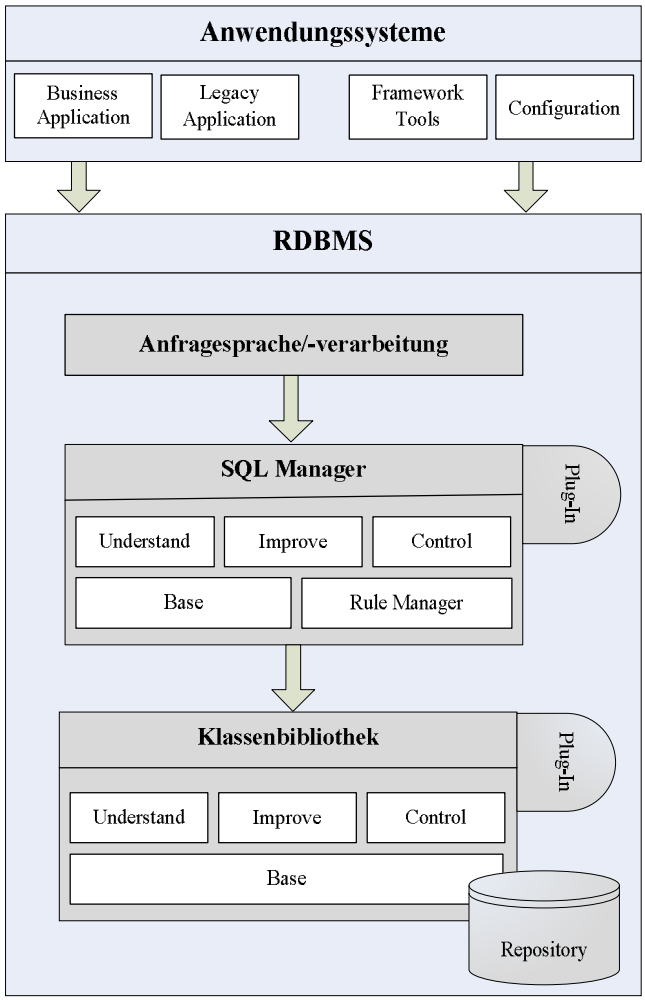
Konzeptionelle Möglichkeiten hierfür bestehen im Einsatz von Plug-ins, der Parametrisierung und Verwendung unterschiedlicher Konfigurationen sowie der Registrierung neuer Komponenten für eine spezifische Aufgabe. Eine flexible Konfiguration und Registrierung von Komponenten innerhalb eines Frameworks bieten vor allem Programmiersprachen, die Reflexion bzw. Introspektion erlauben (Java, .NET-Sprachen). Komponenten in Form von Plug-ins werden dabei durch das Framework registriert, gesteuert und verwendet, d.h. der

[462] Reussner, R.; Hasselbring, W.: Handbuch der Software-Architektur; dpunkt Verlag; 2006; Heidelberg; S. 68
[463] Balzert, H.: Lehrbuch der Software-Technik - Software-Entwicklung, 2. Auflage; Spektrum Akademischer Verlag; Heidelberg, Berlin; 2000; S. 843ff
[464] Reussner, R.; Hasselbring, W.: Handbuch der Software-Architektur; dpunkt Verlag; 2006; Heidelberg; S. 396f

Kontrollfluss wird vom Framework übernommen. Die Architektur einer Anwendung wird damit vom Framework festgelegt.[465]

In Kapitel 4 wurde ein Architekturansatz beschrieben, der die bestehende Basisarchitektur eines RDBMS erweitert. Dieser Ansatz besteht aus zwei wesentlichen Hauptkomponenten, dem „SQL Manager“ und der Klassenbibliothek.

Abb. 31 Architektur der Datenqualitätskomponente



Quelle: eigene Darstellung

Zum einen wird eine Hauptkomponente benötigt, die als Schnittstelle die SQL-Routinen usw. bereitstellt, mit denen die Datenqualität über das RDBMS verbessert bzw. überprüft werden kann. Dieser sogenannte „SQL Manager“ stellt die Verbindung zwischen dem RDBMS und der zweiten Hauptkomponente, einer „Klassenbibliothek“, her, die die eigentlichen Verfahren

[465] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: Entwurfsmuster; Addison-Wesley; München; 1996, S. 31

zur Datenqualität enthält (siehe Abb. 31). Der „SQL Manager“ verwendet dazu SQL-Funktionen, SQL-Prozeduren und Trigger, wie sie im SQL:2003 Standard definiert sind.

Datenqualitätsaspekte werden dabei entweder über Aufrufe der Anwendungssysteme in SQL (Business Application) oder über Konfigurationseinstellungen, die im Repository der Datenqualitätskomponente gespeichert sind, umgesetzt. Dadurch ist es möglich, bestimmte Qualitätsaspekte auch für Legacy Anwendungen zu aktivieren, ohne dass diese geändert werden müssen.

Der „SQL Manager“ enthält fünf Subkomponenten: Neben den vier Komponenten „Base“ für die Basisverfahren, „Understand“ für Verfahren zum Verstehen der Daten, „Improve“ zum Verbessern der Datenqualität, sowie „Control“ zum Steuern und Überwachen der Datenqualität ist vor allem der „Rule Manager“ von entscheidender Bedeutung. Über den „Rule Manager“ werden Geschäftsregeln verwaltet, die die Grundlage für das Steuern und Verbessern der Daten bilden. Verfahren aus den anderen Subkomponenten werden zur Regelbildung verwendet. Über die Regelbildung ist es möglich, Datenprobleme zu protokollieren oder zu vermeiden, ohne dass die auf der Datenbank aufsetzenden Anwendungssysteme geändert werden müssen.

Im Folgenden wird zunächst die Klassenbibliothek als zweite Hauptkomponente beschrieben. Anschließend werden die zwei wichtigen Subkomponenten „Rule Manager“ und „Deduplication“ erläutert. Zentral zur Verbesserung und Steuerung von Datenqualität sind Regeln , die auf bestimmte Ereignisse reagieren und vom „Rule Manager“ verwaltet werden. Über die Subkomponente „Deduplication“ werden Duplikate in Tabellen erkannt und miteinander fusioniert.

5.2 Klassenbibliothek

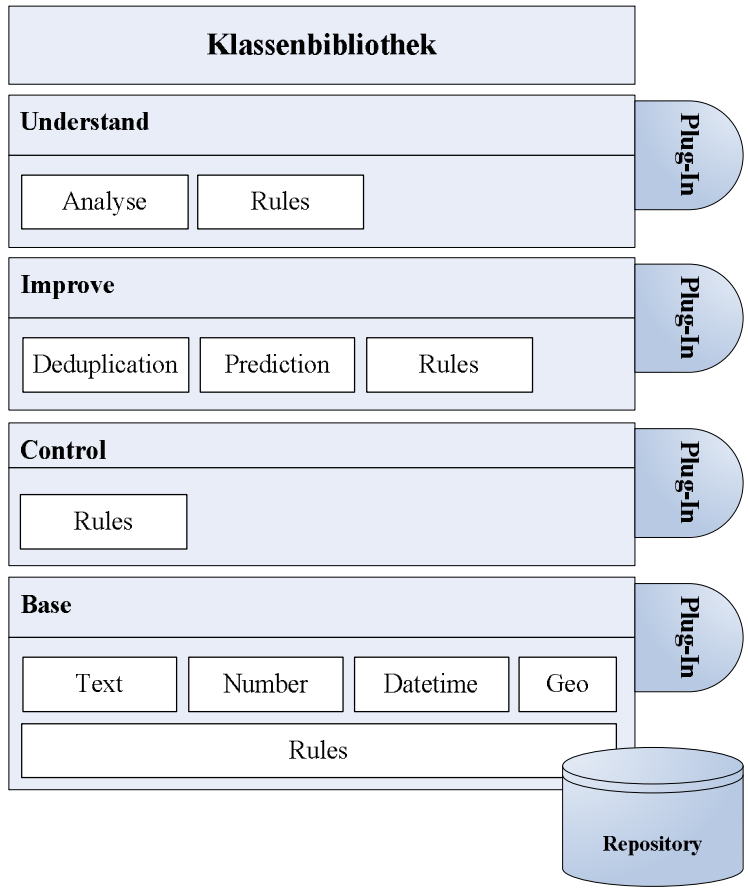
Die zweite Hauptkomponente der Datenqualitätskomponente ist die Klassenbibliothek. Diese enthält die eigentlichen Datenqualitätsverfahren. Der „SQL Manager“ dient dabei hauptsächlich als Hülle zur Verwendung dieser Verfahren. Allgemein lässt sich eine Klassenbibliothek

definieren als eine organisierte selbständige Sammlung von Softwarekomponenten (hier Klassen) als Voraussetzung zur Wiederverwendung von Code „im Kleinen“.[466]

Die Klassenbibliothek besteht aus vier Subkomponenten, die wiederum weitere Subkomponenten enthalten:

- Understand
- Improve
- Control
- Base

Abb. 32 Klassenbibliothek



Quelle: eigene Darstellung

[466] Balzert, H.: Lehrbuch der Software-Technik - Software-Entwicklung, 2. Auflage; Spektrum Akademischer Verlag; Heidelberg, Berlin; 2000; S. 840

Die Basiskomponente enthält grundlegende Encoder-, Matching- und Tokenizer-Verfahren. Über den „SQL Manager“ werden diese direkt als SQL-Routinen aufgerufen. Neben den Subkomponenten „Text“, „Number“, „Datetime“, „Geo“ zur Anwendung von Encoder-, Matching- und Tokenizer-Verfahren enthält die Basiskomponente die Subkomponente „Rules“ zur Verwaltung von Regeln. Diese verwaltet die Regeln persistent im Repository und wird von der Subkomponente „Rule Manager“ des „SQL Managers“ verwendet, um Regeln zu verwalten, zu aktivieren und auszuführen. Änderungen an der Basis oder an der Datenstruktur des Repository führen damit nicht zwangsläufig zu Änderungen an den Komponenten „Understand“, „Improve“ und „Control“, die die Basiskomponente verwenden.

Über SQL-Routinen im „SQL Manager“ können grundlegende Basisverfahren wie z.B. Encoder, Matching, Tokenizer aufgerufen werden. Eigene Komponenten für diese Verfahren können über den „SQL Manager“ registriert und über eine generische SQL-Funktion verwendet werden.

Das Steuern und Überwachen der Datenqualität erfolgt über die Definition von Regeln. Werden über den „Rule Manager“ die Geschäftsregeln verwaltet, so wird über die Subkomponente „Control“ die Einhaltung dieser gemessen und überwacht.

Die Komponente „Improve“ besteht aus den Subkomponenten „Deduplication“, „Prediction“ und „Rules“ und dient zum konkreten Anpassen der Daten zur Verbesserung der Datenqualität. Über die Komponente „Deduplication“ werden Verfahren zur Duplikaterkennung konfiguriert, die dann innerhalb einer Geschäftsregel oder zur einmaligen Bereinigung von Duplikaten verwendet werden. Mit der Komponente „Prediction“ können Daten standardisiert, korrigiert oder angereichert werden. Ein Predictor kann dabei aus einem maschinellen Lernverfahren bestehen oder aber Daten z.B. aus Nachschlagetabellen ermitteln. Über die Subkomponente „Rules“ können Daten schließlich automatisch standardisiert, korrigiert oder angereichert werden, indem eine Regel automatisch bei der Veränderung von Daten ausgeführt wird.

Schließlich besteht die Komponente „Understand“ der Klassenbibliothek aus den beiden Subkomponenten „Analyse“ und „Rules“. „Analyse“ enthält dabei Verfahren zur Erkennung von Primärschlüsseln, Fremdschlüsselbeziehungen und zur Ermittlung von Schema- und

Dateneigenschaften. Über die Subkomponente „Rules“ können Regeln aus den Daten über Verfahren wie z.B. maschinelle Lernverfahren extrahiert werden.

Letztendlich können Verfahren aus den Subkomponenten „Base“ und „Improve“ natürlich auch zum Verstehen der Daten verwendet werden. Ein Predictor kann z.B. dazu dienen, Unterschiede in den vorhergesagten und den vorhandenen Daten zu analysieren. Über Encoder können Häufigkeitsverteilungen von Codes überprüft werden usw.

5.3 Regeln

Zur Verbesserung der Datenqualität spielt die Komponente zur Bearbeitung und Anwendung von Geschäftsregeln eine wesentliche Rolle. Neben der Definition von Geschäftsregeln unterstützt die Komponente verschiedene Ereignisse, um die Regelanwendung auszulösen. Zur Analyse der Regeln und zur Auswertung dieser sollte eine administrative Komponente die Möglichkeit bieten, Reports zu verwalten und die Anwendung und Verletzung von Regeln selbst auszuwerten. Zur Definition einer Regel stellen die Elemente Bezeichnung, Regelnummer, Versionsnummer, Regeltyp, Beschreibung der Regel und der verwendeten Datenbankobjekte, Bemerkungen, Anwendbarkeit (Aktiv / Inaktiv / Protokollierung), Datum der Erstellung/Änderung sinnvolle Inhalte dar.

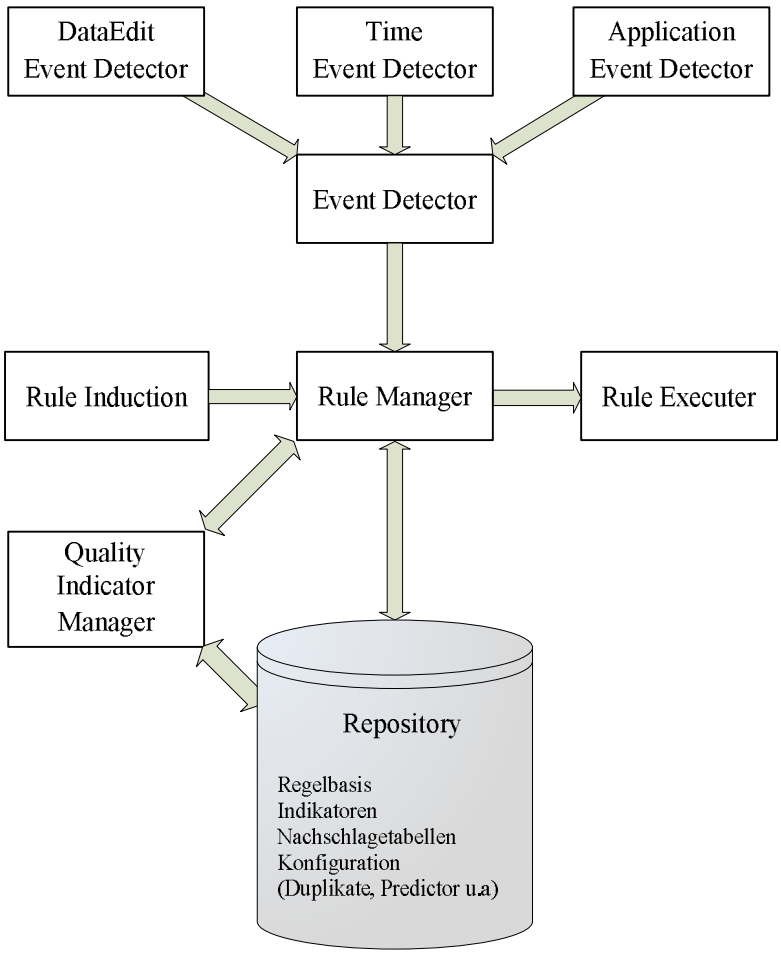
Der „Rule Manager“ verwaltet die Geschäftsregeln (siehe Abb. 33), die entweder manuell oder über Regelinduktion („Rule Induction“) erfasst wurden. Über den „Event Detector“ werden drei unterschiedliche Ereignisse erkannt:

- DataEdit Event (Manipulation von Daten)
- Time Event (zeitliche Ereignisse)
- Application Event (anwendungsbezogene Ereignisse)

Der „Event Detector“ verwaltet die einzelnen Ereignisse und gibt bei deren Auslösung diese an den „Rule Manager“ weiter. Abhängig davon, ob die Bedingung einer Regel eintritt, wird die Aktion der Regel über den „Rule Executer“ abgearbeitet. Über den „Quality Indicator Manager“ wird die Einhaltung der einzelnen Regeln überwacht und die Messung einer

Datenqualitätsdimension zu verschiedenen Zeitpunkten gespeichert, um einen Trend der entsprechenden Datenqualitätsdimension zu erkennen.

Abb. 33 Rule Manager



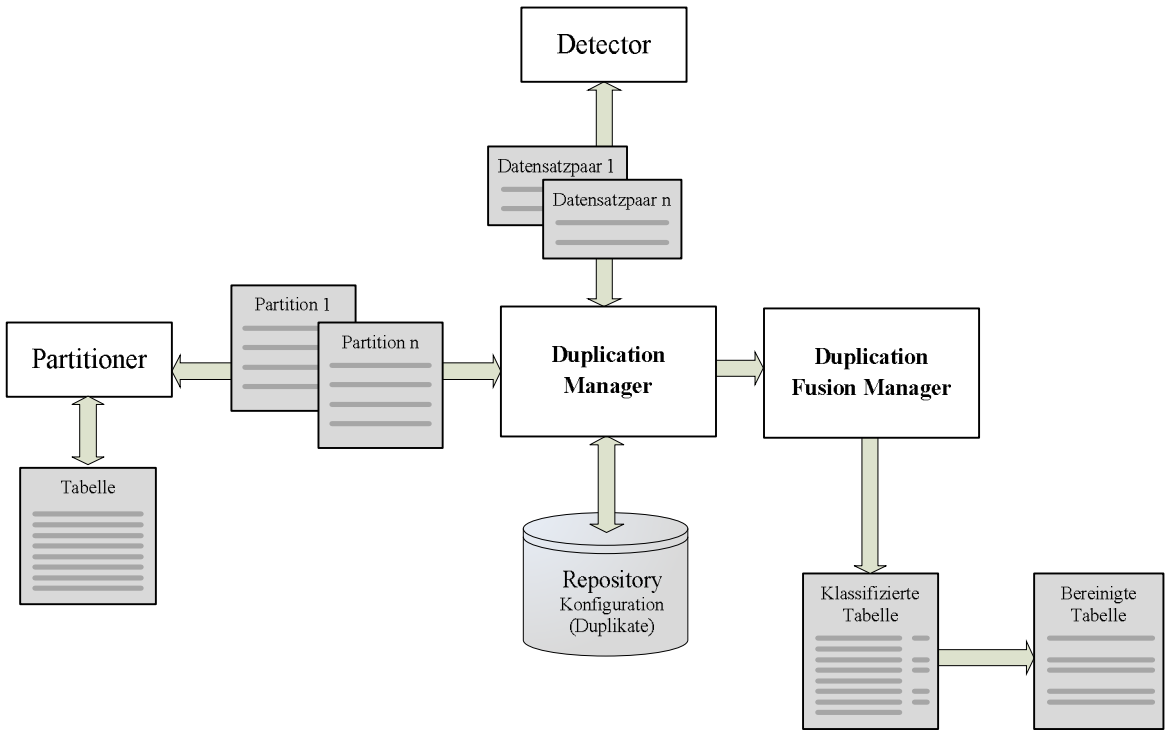
Quelle: eigene Darstellung

Im Repository der Datenqualitätskomponente werden die Geschäftsregeln und die Datenqualitätsindikatoren persistent gespeichert. Daneben enthält das Repository Konfigurationsdaten (z.B. zur Duplikaterkennung) und Nachschlagetabellen zur Standardisierung, Korrektur oder Datenanreicherung.

5.4 Duplikate

Die Erkennung von Duplikaten besteht aus drei wesentlichen Teilkomponenten. Über den „Duplication Manager“ wird die Duplikaterkennung gesteuert. Der „Partitioner“ partitioniert einen Datenbestand in einzelne Blöcke und reduziert dadurch den Suchraum. Über den „Detector“ wird für ein vom „Partitioner“ gelesenes Datensatzpaar ermittelt, ob es sich um Duplikate handelt.

Abb. 34 Deduplication



Quelle: eigene Darstellung

Zur Duplikaterkennung initiiert der „Duplication Manager“ beim „Partitioner“ die Einteilung der Tabelle in einzelne Partitionen. Aus diesen werden vom „Partitioner“ Datensatzpaare gebildet, die über den „Duplication Manager“ an den „Detector“ weitergeleitet werden. Dieser entscheidet schließlich auf Basis eines probabilistischen oder deterministischen Verfahrens, ob es sich bei dem Datensatzpaar um Duplikate handelt. Ist das der Fall, so werden die beiden Datensätze als solche gekennzeichnet.

Nachdem alle vom „Partitioner“ ermittelten Datensatzpaare klassifiziert wurden, überprüft der „Duplication Manager“ auf transitive Duplikate und ändert gegebenenfalls noch einmal die Duplikatkennung. Sind nun alle Duplikate ermittelt, kann in einem letzten Schritt über den „Duplication Fusion Manager“ eine Verschmelzung der als Duplikate erkannten Datensätze zu einem einzigen neuen Datensatz erfolgen.

6. Implementation eines Prototyps

6.1 Allgemein

Nachdem in Kapitel 5 die Architektur zur Einbettung einer Datenqualitätskomponente aufgezeigt wurde, folgt nun die Beschreibung einer konkreten Implementation, in der die Architektur zur Einbettung aus Kapitel 4 und die in Kapitel 3 beschriebenen Verfahren und Algorithmen Berücksichtigung finden.

Da kommerzielle RDBMS den SQL:2003 Standard bei weitem noch nicht vollständig umgesetzt haben,[467] wurden im Prototypen vor allem Sprachkonstrukte aus den bekannten vier RDBMS (IBM DB2, Oracle, Informix, Microsoft SQL Server) verwendet, die sich weitgehend an den Standard anlehnen. Zu diesen Sprachkonstrukten zählen SQL-Funktionen[468], SQL-Prozeduren und Trigger. Zusätzlich wurden bestimmte Verfahren mit benutzerdefinierten Aggregatfunktionen gelöst, die zwar im SQL-Standard noch nicht enthalten, aber von den meisten bekannten RDBMS wie z.B. dem Informix Dynamics Server, Microsoft SQL Server, Oracle, MySQL und PostgreSQL unterstützt werden. Für die effiziente Nutzung der Datenqualitätskomponente sind diese jedoch nicht zwingend notwendig, erleichtern aber die Verwendung.

Als Umgebung wurde für das relationale Datenbankmanagementsystem der Microsoft SQL Server 2005 in Verbindung mit der .NET CLR-Integration gewählt, da in .NET CLR verschiedene Programmiersprachen parallel in einem System verwendet werden können. Mehrere der in Kapitel 3 vorgestellten originalen Algorithmen wurden in den Programmiersprachen Java, C und C++ implementiert. Durch die Verwendung von .NET CLR wurde deren Integration in die Datenqualitätskomponente als Plug-in wesentlich erleichtert.

[467] Türker, C.: SQL:1999 & SQL:2003 - Objektrelationales SQL, SQLJ & SQL/XML; dpunkt Verlag; 2003; S. 460

[468] Hierzu gehören auch die seit SQL:2003 standardisierten Tabellenrückgabefunktionen.

Alle Verfahren werden in einer eigenen Datenbank gespeichert, die aus mehreren Schemata besteht. Hierzu gehören:

- Text
- Number
- Date
- Distance
- Rules
- Understand
- Improve
- Control
- Tools

Die Basisverfahren sind in den Schemata „Text“, „Number“, „Date „und „Distance“ gespeichert. Verfahren zum Verstehen der Daten befinden sich im Schema „Understand“, zum Verbessern in „Improve“ und zur Steuerung und Überwachung der Datenqualität in „Control“.

Zusätzlich enthält das Schema „Rules“ alle Verfahren, die mit Regeln zusammenhängen und die sowohl zum Verstehen, Verbessern und Steuern verwendet werden können. Das Schema „Tools“ enthält schließlich allgemeine Routinen wie z.B. Registrieren eines Plug-in, Protokollieren, Benachrichtigen bei Regelverletzungen u.ä.

Außerdem existieren drei weitere Schemata zum persistenten Speichern von Metadaten und Nachschlagetabellen:

- Repository
- Deduplication
- Lookup

In den folgenden Abschnitten werden alle Klassen und Erweiterungsmöglichkeiten sowie die exemplarische Anwendung einzelner Methoden über SQL erklärt. Eine ausführliche Beschreibung der SQL-Routinen mit einer Auflistung benannter Parameter ist in Anhang A und B zu finden.

6.2 Basis

6.2.1 Allgemein

Die Basisverfahren bestehen aus Verfahren zum Erzeugen von Codes aus einem Wert (Encoder), zum Überprüfen der Ähnlichkeit zweier Werte (Matching) und zum Aufteilen eines Wertes in Teilwerte wie z.B. linguistische Einheiten oder N-Grams (Tokenizer). Da diese drei Verfahrenstypen vom Datentyp (Text, Number, Date, Distance) abhängig sind, werden sie den jeweiligen Schemata für diesen Datentyp untergeordnet. Für die Überprüfung, ob zwei Zahlen oder zwei Zeichenketten ähnlich sind, werden z.B. weitgehend unterschiedliche Algorithmen verwendet (siehe Kapitel 3).

Für Matching, Encoder und Tokenizer existiert jeweils eine generische Funktion, die ein beliebiges Matching-, Encoder- oder Tokenizer-Verfahren aufrufen kann. Dazu werden an diese generische Funktion neben den spezifischen Parametern der vollständige Klassenname des zu verwendenden Verfahrens, die Bibliothek (Assembly) und benannte Parameter übergeben. Diese Vorgehensweise findet sich nicht nur in der Basis, sondern durchgängig im gesamten Framework. Dadurch ist es möglich, eigene Bibliotheken, die als Plug-in bei der Datenqualitätskomponente registiert wurden, aufzurufen. Diese Vorgehensweise wird in den folgenden Abschnitten zu Erweiterungen jeweils detailliert erläutert.

6.2.2 Encoder

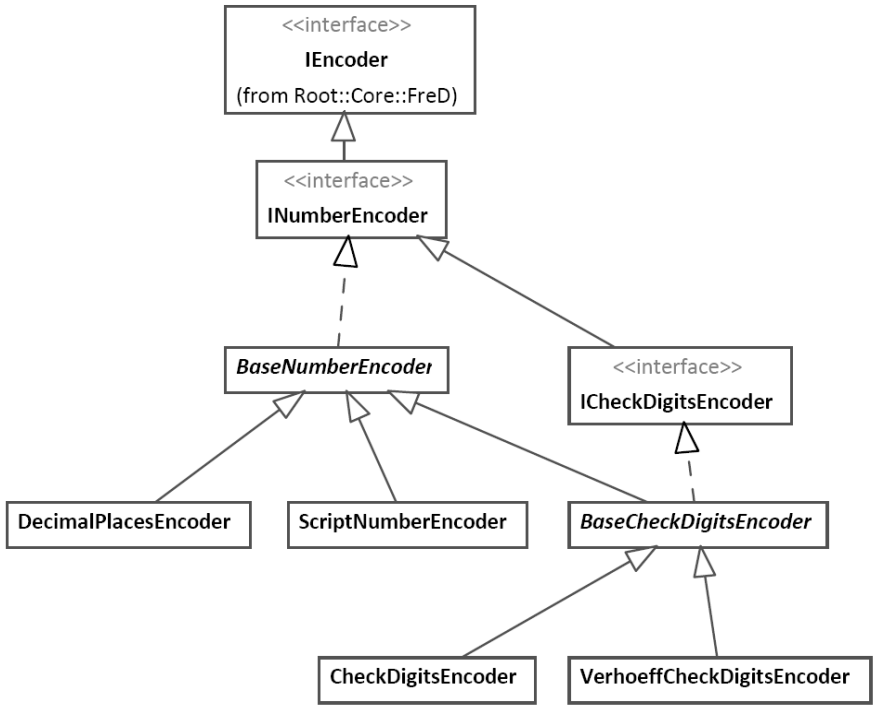
Bei einem Encoder handelt es sich um einen Kodierer, der aus einem Wert einen Code erzeugt. Algorithmen zur Erzeugung von Codes wurden in Abschnitt 3.2.1 ausführlich beschrieben.

Der Namensraum *FreD.Number.Encoder* enthält alle in der Klassenbibliothek implementierten Kodierer für Zahlen. Ein Kodierer für Zahlen muss die Schnittstelle *INumberEncoder* implementieren, die wiederum von der allgemeinen Schnittstelle *IEncoder* abgeleitet ist. Diese Schnittstelle definiert die Methode *Encode*, der eine Zahl zum Kodieren übergeben wird.

Über die Klasse *DecimalPlacesEncoder* wird für eine Zahl die Anzahl der Nachkommastellen ermittelt. *CheckDigitsEncoder* und *VerhoeffCheckDigitsEncoder* erzeugen eine Prüfziffer und implementieren zusätzlich über ihre Basisklasse *BaseCheckDigitsEncoder* die Methode *Decode* zur Überprüfung einer Prüfziffer.

Der Klasse *ScriptNumberEncoder* kann Quellcode in einer .NET Programmiersprache übergeben werden, die einen Kodierer für Zahlen implementiert. Auf diese Möglichkeit wird ausführlich in Abschnitt 6.2.5 eingegangen.

Abb. 35 FreD.Number.Encoder



Quelle: eigene Darstellung

Um diese Klassen als SQL-Funktionen im RDBMS zur Verfügung zu stellen, existiert im Namensraum *FreD.Number.Encoder* eine dazugehörige Wrapperklasse. Diese enthält zudem die beiden SQL-Funktion *Encode* und *Decode* zum generischen Aufruf eines beliebigen Kodierers. Neben der Zahl, die kodiert bzw. dekodiert werden soll, wird *Encode* und *Decode* dabei der vollständige Klassenname, der Bibliotheksname und eventuell benannte Parameter übergeben.

Die Bereitstellung im RDBMS erfolgt dann über die bekannten SQL-Sprachkonstrukte zum Anlegen benutzerdefinierter Funktionen.

Die folgende SQL-Anweisung zeigt die Verwendung im RDBMS über die generische Encoder-Funktion:

```
SELECT CardType, CardNumber,
       q.Number.Encode( CardNumber,
                        'FreD.Number.Encoder.VerhoeffCheckDigitsEncoder',
                        'Core', null )
FROM   Sales.CreditCard
```

Entsprechend sieht der spezifische Encoder-Aufruf folgendermaßen aus:

```
SELECT CardType, CardNumber,
       q.Number.CheckDigitsVerhoeffEncode(CardNumber)
FROM   Sales.CreditCard
```

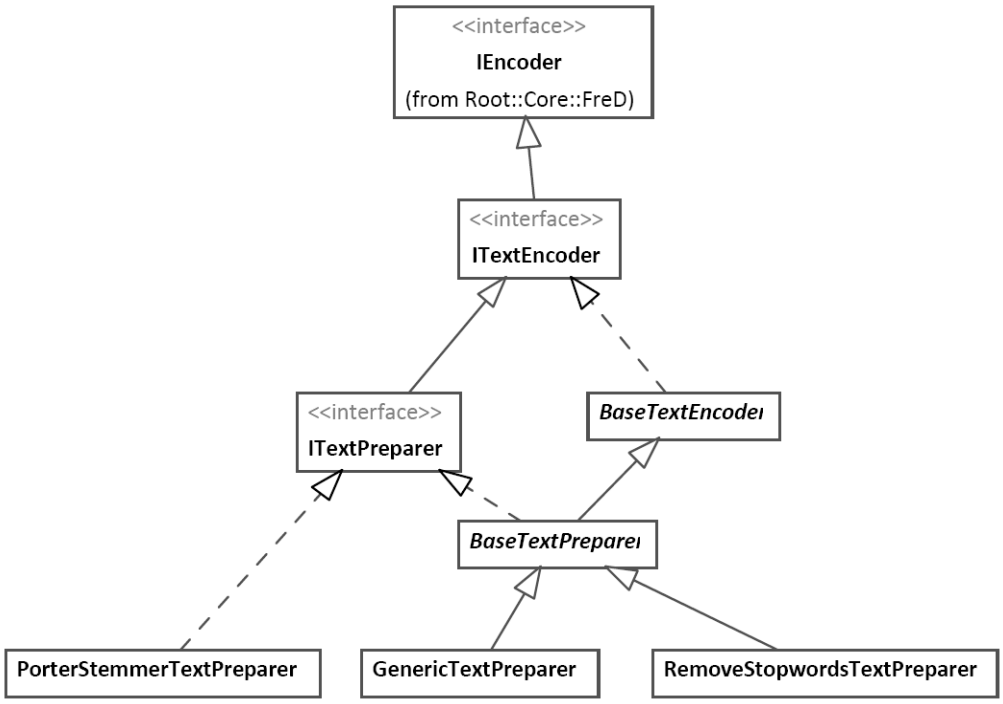
Im Gegensatz zum generischen können beim spezifischen Aufruf zugunsten der einfacheren Anwendung keine zusätzlichen benannten Parameter übergeben werden.

Der Namensraum *FreD.Text.Encoder* enthält alle in der Klassenbibliothek implementierten Kodierer für Text bzw. alphanumerische Daten. Hierbei kann unterschieden werden zwischen Kodierverfahren zur Bildung eines Hashwertes (Hashing), zur Musterbildung (Pattern), zur Erzeugung phonetischer Codes und zur Vorbereitung von Text (Preparer). Ein Kodierer für Text implementiert die Schnittstelle *ITextEncoder*, die genau wie bei numerischen Encodern von der allgemeinen Schnittstelle *IEncoder* abgeleitet ist.

Preparer (siehe Abb. 36) implementieren die Schnittstelle *ITextPreparer*. In der Klassenbibliothek sind zwei Preparer implementiert: Der *GenericTextPreparer* (zur allgemeinen Umwandlung von Text, Groß-/Klein, Zeichen ignorieren u.ä.) und der *RemoveStopwordsTextPreparer*, der aus einer ihm übergebenen Zeichenkette Stoppwörter aus einer im Repository gespeicherten Liste ausfiltert. Daneben existieren als Plug-ins verschie-

dene Stemmingalgorithmen (z.B. der *PorterStemmerTextPreparer*[469]), die direkt von *ITextPreparer* abgeleitet sind.

Abb. 36 FreD.Text.Encoder (Preparer)



Quelle: eigene Darstellung

Wie bei numerischen Kodierern existiert auch hier eine Wrapperklasse, die die beschriebenen Klassen in SQL als Funktionen bereitstellt.

Die folgende SQL-Anweisung zeigt zunächst einen generischen Aufruf, der das Plug-in zum Porter-Stemming aufruft.

```
SELECT q.Text.Encode( DocumentSummary,
                      'FreD.Text.Encoder.PorterStemmerTextPreparer',
                      'PorterStemmerTextPreparer', null )
FROM   Production.Document
```

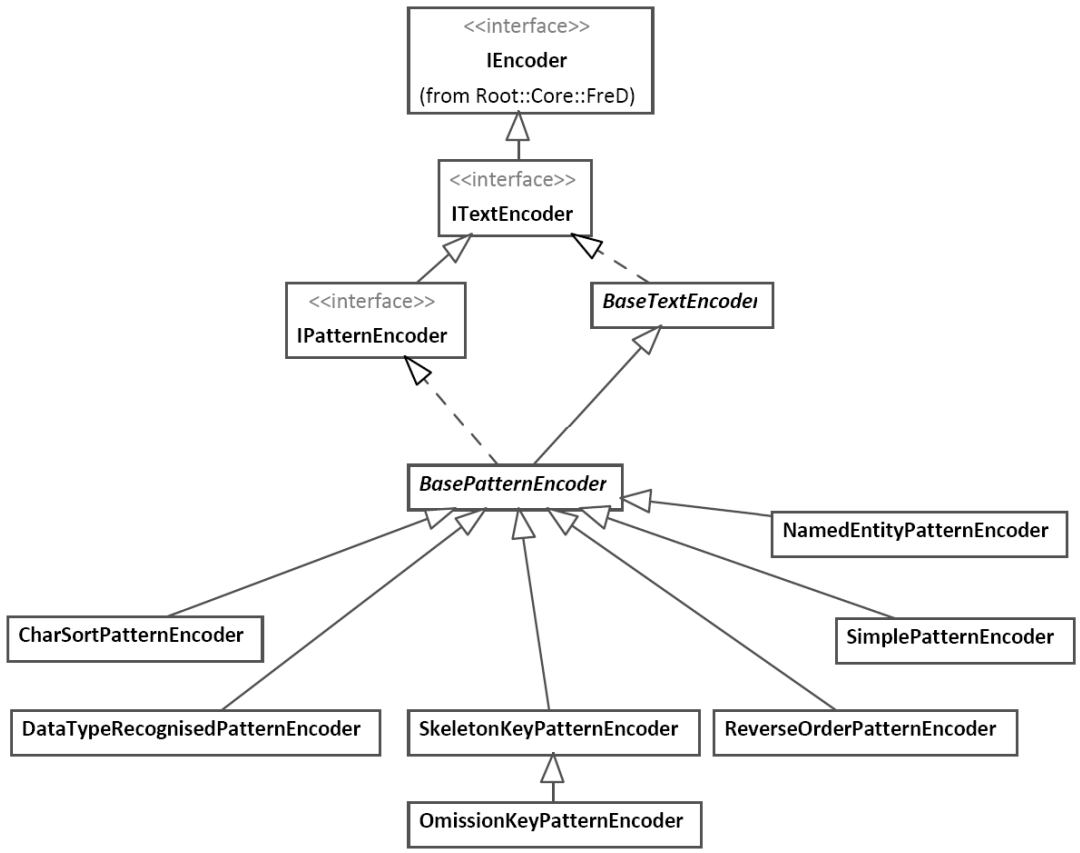
Ein spezifischer Aufruf zum Entfernen von Stoppwörtern sieht wie folgt aus.

[469] Als Plug-In's sind außerdem die Stemming-Algorithmen von Lovins, Paice/Husk, Krovetz, Caumann eingebunden und der UEA-Stemming-Algorithmus

```
SELECT q.Text.PreparerRemoveStopwords(DocumentSummary)
FROM   Production.Document
```

Verfahren zur Erzeugung von Pattern (siehe Abb. 37) implementieren die *IPatternEncoder*-Schnittstelle.

Abb. 37 FreD.Text.Encoder (Pattern)



Quelle: eigene Darstellung

Neben der in der Klassenbibliothek enthaltenen *SimplePatternEncoder*-Klasse, für die ein gemeinsames Zeichen für Zahlen, Großbuchstaben, Kleinbuchstaben und andere Zeichen festgelegt wird, existieren in der Klassenbibliothek noch sechs weitere Pattern-Kodierer. Der *CharSortPatternEncoder* sortiert eine Zeichenkette alphabetisch. Der *DataTypeRecognisedPatternEncoder* kann verwendet werden, um den eigentlichen Datentyp eines Attributes zu ermitteln, indem die Klasse den Datentyp für eine Zeichenkette erkennt und diesen als Datentypbeschreibung zurückliefert. Über *ReverseOrderPatternEncoder* werden die übergebenen Zeichen in umgekehrter Reihenfolge zurückgeliefert. Dies kann z.B. bei der Duplikaterkennung sinnvoll sein, wenn eine Zeichenkette aus mehreren unterschied-

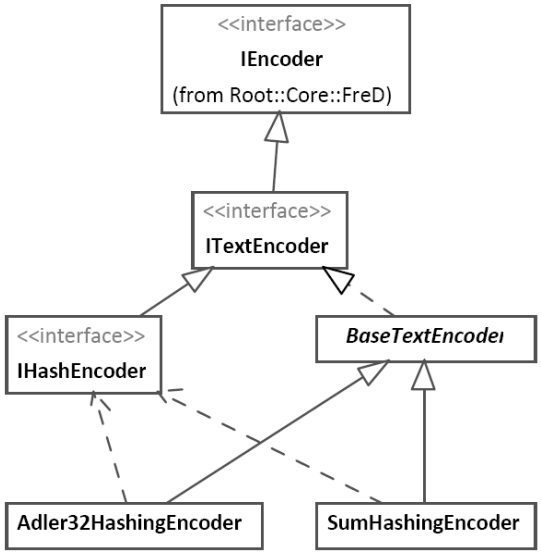
lichen semantischen Werten besteht. *NamedEntityPatternEncoder* liefert für einen übergebenen Wert den semantischen Typ zurück und kann z.B. zum Parsing von Spalten mit mehreren Werten eingesetzt werden. *SkeletonKeyPatternEncoder* und *OmissionKeyEncoder* schließlich werden vorrangig zur Erkennung von Eingabefehlern verwendet.

Die folgende SQL-Anweisung zeigt die spezifischen Aufrufe zu den beschriebenen Patternklassen:

```
SELECT Lastname, Phone,
       q.Text.PatternsSimple( Phone ),
       q.Text.PatternsCharSort( Phone ),
       q.Text.PatternsReverseOrder( Phone ),
       q.Text.PatternsOmissionKey( Lastname ),
       q.Text.PatternsSkeletonKey( Lastname ),
       q.Text.PatternsDataTypeRecognised( Phone )
       q.Text.PatternsNamedEntity( Lastname )
FROM   Person.Contact
```

Als Hashverfahren (siehe Abb. 38) sind in der Bibliothek der Adler32-Algorithmus und die einfache Quersummenbildung implementiert. Hashverfahren implementieren direkt die Schnittstelle *IHashEncoder*.

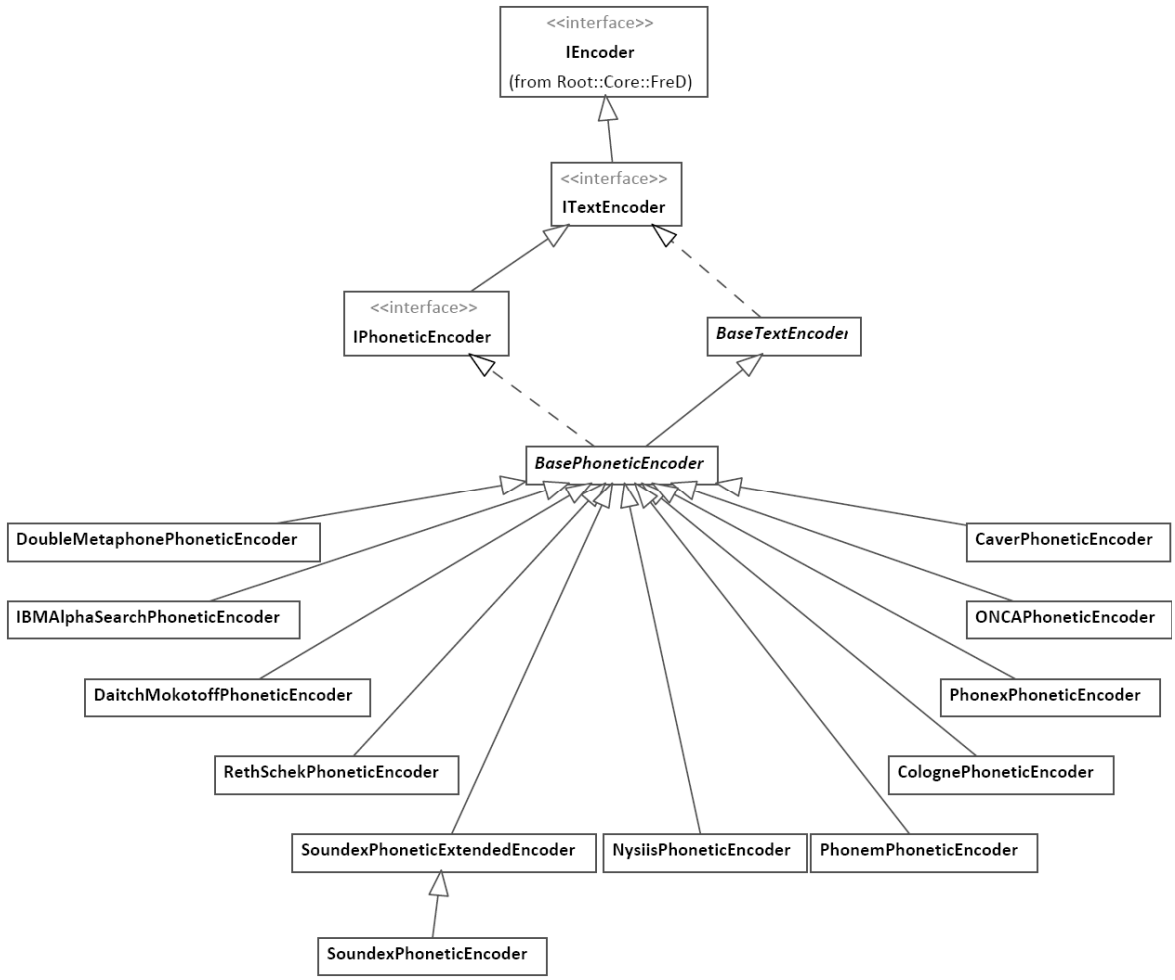
Abb. 38 FreD.Text.Encoder (Hashing)



Quelle: eigene Darstellung

Algorithmen zur Erzeugung phonetischer Codes (siehe Abb. 39) müssen die Schnittstelle *IPhoneticEncoder* implementieren. Neben dem bekannten phonetischen Soundex-Code, der über die Klasse *SoundexPhoneticEncoder* implementiert ist, sind die meisten der in Abschnitt 3.2.1.3 beschriebenen Algorithmen implementiert. Der von J. Michael entwickelte phonet-Code ist nicht in der Klassenbibliothek, sondern als Plug-in enthalten, da der originale C-Code aus dem Jahr 1992 verwendet wurde.[470] Die Klasse *SoundexPhoneticEncoder* ist von der *SoundexPhoneticExtendedEncoder* abgeleitet, die ein eigenes Mapping für die einzelnen alphabetischen Zeichen zu einem gemeinsamen Codezeichen erlaubt.

Abb. 39 FreD.Text.Encoder (Phonetic)



Quelle: eigene Darstellung

[470] Siehe zum Quellcode ftp://ftp.heise.de/pub/ct/listings/phonet.zip

Die folgende SQL-Anweisung beinhaltet einen generischen Aufruf, um ein eigenes Mapping für den Soundex-Code zu verwenden, der allen Vokalen das Zeichen ‚X' zuweist:

```
SELECT LastName,
       q.Text.Encode( LastName,
       'FreD.Text.Encoder.SoundexPhoneticEncoder', 'Core',
       'Mapping=X123X12-X22455X12623X1-2-2' )
FROM   Person.Contact
```

Den Aufruf spezifischer phonetischer Encoder zeigt die folgende SQL-Anweisung:

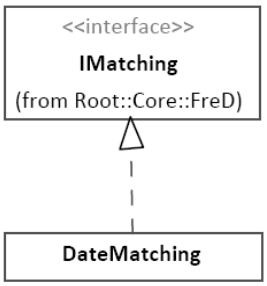
```
SELECT LastName,
         q.Text.PhoneticCaver( LastName ),
         q.Text.PhoneticCologne( LastName ),
         q.Text.PhoneticDaitchMokotoff( LastName ),
         q.Text.PhoneticDoubleMetaphone( LastName ),
         q.Text.PhoneticIBMAlphaSearch( LastName ),
         q.Text.PhoneticNysiis( LastName ),
         q.Text.PhoneticONCA( LastName ),
         q.Text.PhoneticPhonem( LastName ),
         q.Text.PhoneticPhonex( LastName ),
         q.Text.PhoneticRethSchek( LastName ),
         q.Text.PhoneticSoundex( LastName ),
         q.Text.PhoneticGerman( LastName )
FROM Person.Contact
```

6.2.3 Matching

Beim Matching wird die Ähnlichkeit zwischen zwei Werten über ein Proximitätsmaß ausgedrückt. Genau wie bei der Erzeugung von Codes gibt es je nach Datentyp unterschiedliche Algorithmen. Im Folgenden werden die Algorithmen zur Überprüfung der Ähnlichkeit bei den Datentypen Date, Distance, Number und Text beschrieben.

Für den Datentyp Date existiert in der Klassenbibliothek eine einfache Klasse zur Überprüfung der Ähnlichkeit zweier Datumswerte, indem ein maximaler Differenzwert festgelegt wird. Die Klasse *DateMatching* ist direkt von *IMatching* abgeleitet und implementiert die Schnittstellenmethode *Match*.

Abb. 40 FreD.Date.Matching



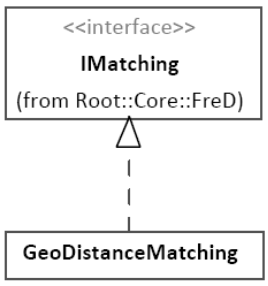
Quelle: eigene Darstellung

Die folgende SQL-Anweisung vergleicht über einen generischen Aufruf zwei Datumswerte und errechnet deren Ähnlichkeit auf Basis der Eigenschaft *MaxTimeSpan*.

```
SELECT OrderDate,
       ShipDate,
       q.Date.Match( OrderDate, ShipDate,
                     'FreD.Date.Matching.DateMatching', 'core',
                     'MaxTimeSpan=14.00:00:00' )
FROM   Sales.SalesOrderHeader
```

Über den Datentyp Distance werden Entfernungen zwischen Orten auf Grundlage von Geodaten (Längen-, Breitengrad) errechnet. An die *Match*-Methode der *GeoDistanceMatching*-Klasse werden zwei Postleitzahlen übergeben. Dann wird die Entfernung in Kilometern berechnet und auf Basis einer vom Benutzer angegebenen maximalen Entfernung ein Ähnlichkeitsmaß zurückgeliefert.

Abb. 41 FreD.Distance.Matching



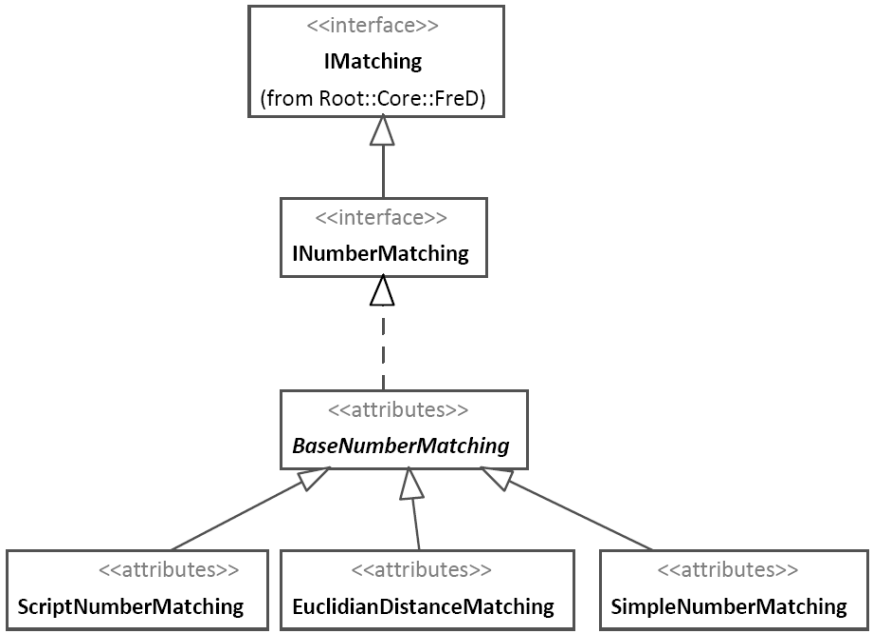
Quelle: eigene Darstellung

Die folgende SQL-Anweisung vergleicht über einen generischen Aufruf zwei Postleitzahlen und errechnet deren Ähnlichkeit auf Basis der Eigenschaft *MaxDistance*.

```
SELECT q.Distance.Match(
          '22880', '20095',
          'FreD.Distance.Matching.GeoDistanceMatching', 'core',
          'Country=DE, MaxDistance=100' )
```

Algorithmen zur Errechnung eines Ähnlichkeitsmaßes von zwei Zahlen sind im Namensraum *FreD.Number.Matching* implementiert.

Abb. 42 FreD.Number.Matching



Quelle: eigene Darstellung

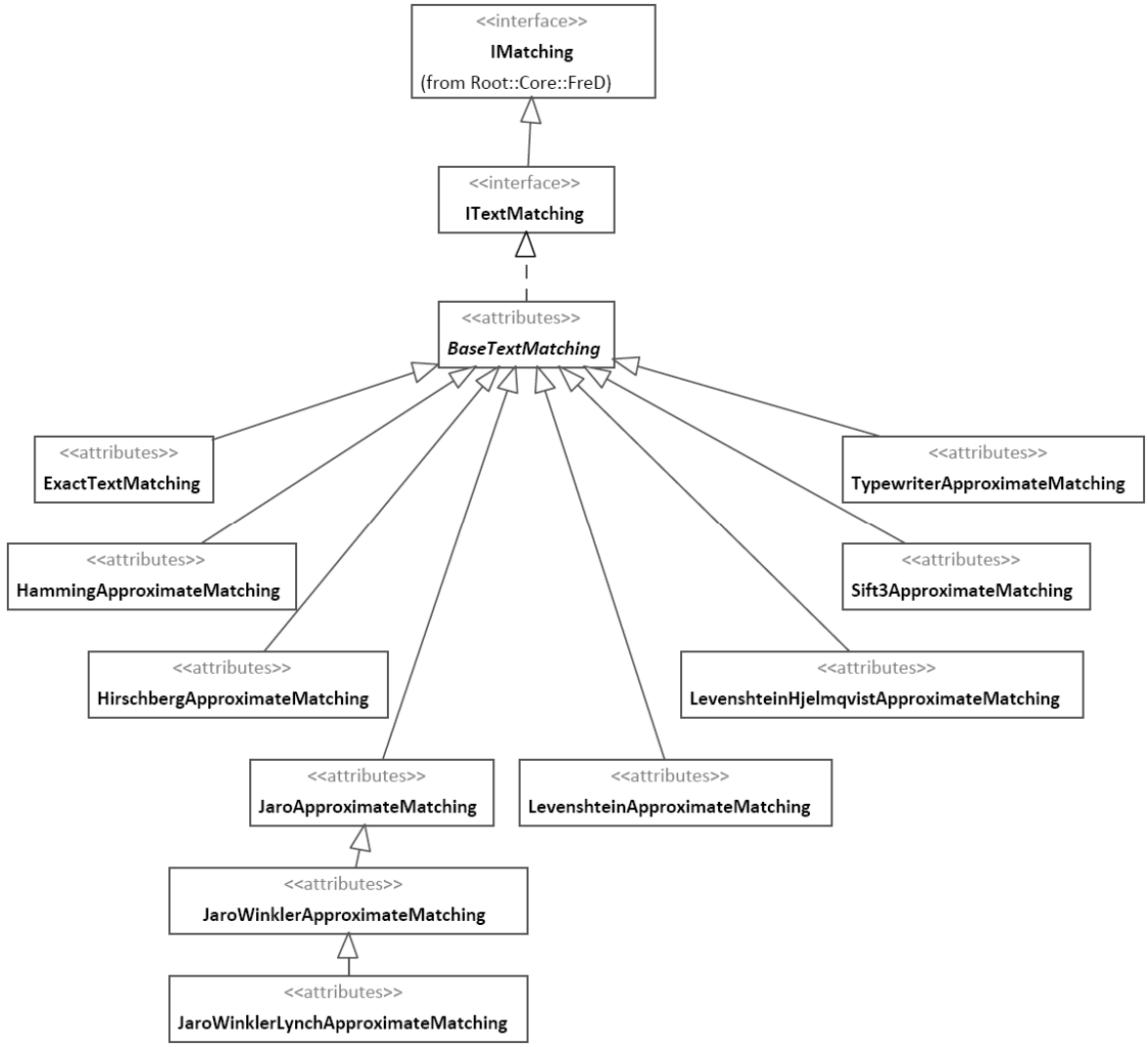
Neben der Klasse *ScriptNumberMatchingEncoder*, mit der eigene Algorithmen als Skript an die *Match*-Methode übergeben werden, existieren Implementierungen zur euklidischen Distanz in *EuclidianDistanceMatching* und eine einfache Berechnung in *SimpleNumberMatching*, dem ein maximaler Distanzwert übergeben wird.

Folgendes Beispiel einer SQL-Anweisung berechnet die euklidische Distanz als Ähnlichkeitsmaß umgerechnet:

```
SELECT StandardCost, ListPrice,
       q.Number.MatchingEuclidian( StandardCost, ListPrice )
FROM   Production.Product
```

Der Namensraum *FreD.Text.Matching* enthält die meisten der in Abschnitt 3.2.2.3 beschriebenen Algorithmen zum Vergleich von Zeichenketten. Bei den Algorithmen kann unterschieden werden zwischen Zeichen- und Token-basierten Verfahren. Abb. 43 zeigt zunächst die Klassenhierarchie für die Zeichen-basierten Algorithmen.

Abb. 43 FreD.Text.Matching (Zeichen-basiert)



Quelle: eigene Darstellung

Die Klasse *ExactTextMatching* überprüft die exakte Übereinstimmung zweier Zeichenketten. Dementsprechend wird als Proximitätsmaß entweder 1 bei Übereinstimmung oder 0 bei

Nicht-Übereinstimmung zurückgegeben. Bis auf den Sift3-Algorithmus[471] sind alle Algorithmen der anderen Klassen in Abschnitt 3.2.2.3 beschrieben.

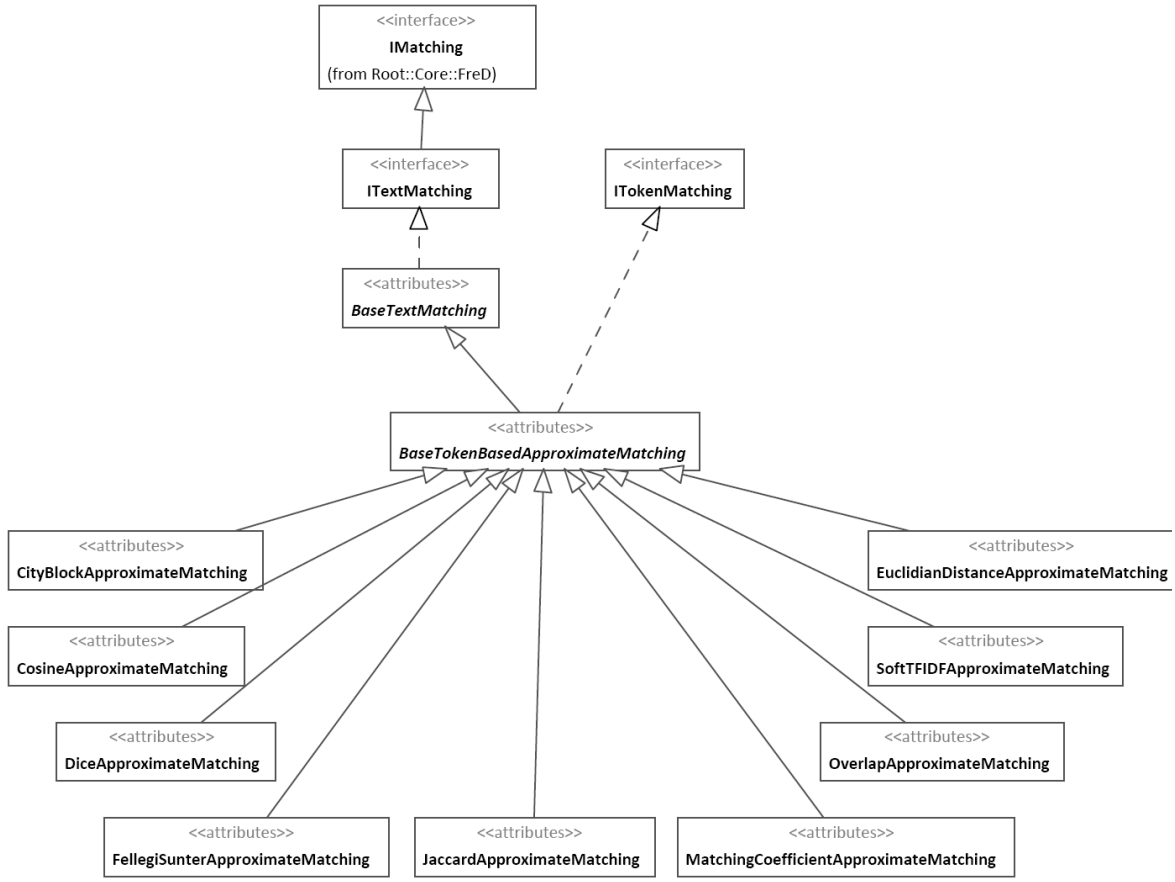
Die folgende SQL-Anweisung verwendet Matching-Routinen zur Ermittlung von Duplikaten in einer Tabelle:

```
SELECT a.LastName, b.LastName,
       a.AddressLine1, b.AddressLine1,
       a.City, b.City,
       a.PostalCode, b.PostalCode
FROM   Employees a, Employees b
WHERE  a.EmployeeId > b.EmployeeId AND
       q.Text.PhoneticSoundex( a.LastName ) =
       q.Text.PhoneticSoundex( b.LastName ) AND
       q.Text.MatchingJaro( a.LastName, b.LastName ) > 0.7 AND
       q.Text.MatchingHirschberg(a.AddressLine1, b.AddressLine1 ) > 0.7 AND
       q.Distance.MatchingSimple(a.PostalCode, b.PostalCode,'US',50) > 0.9
```

In Abb. 44 ist die Klassenhierarchie der Token-basierten Algorithmen zum Vergleich von Zeichenketten dargestellt. Token-basierte Algorithmen implementieren sowohl die Schnittstelle *ITextMatching*, als auch die Schnittstelle *ITokenMatching*. Zu den probabilistischen Matching-Verfahren gehören der Algorithmus zur Duplikaterkennung nach Fellegi und Sunter und der TF-IDF-Algorithmus, die beide in der Klassenbibliothek implementiert sind. Alle anderen implementierten Algorithmen sind deterministische Verfahren. Alle Klassen benötigen zwingend einen Tokenizer, um die einzelnen Zeichenketten, die verglichen werden sollen, in einzelne Token aufzuteilen. Als Standard wird hierbei die Klasse *GenericTokenizer* (siehe nächsten Abschnitt) verwendet, sofern kein Tokenizer definiert wird. Über den benannten Parameter *TokenizerTokenBased* kann jedoch jeder Klasse der Token-basierten Verfahren auch ein beliebiger anderer Tokenizer zugewiesen werden.

[471] Der Sift3-Algorithmus ist ein performanter Algorithmus, der den „Longest Common Substring“ berechnet. Eine Kurzbeschreibung und eine Implementierung hierzu sind unter http://siderite.blogspot.com/2007/04/super-fast-and-accurate-string-distance.html zu finden.

Abb. 44 FreD.Text.Matching (Token-basiert)



Quelle: eigene Darstellung

Die folgende erste SQL-Anweisung berechnet für zwei Zeichenketten das Jaccard-Ähnlichkeitsmaß (zur Trennung der Zeichenkette in Token wird als Standard der *GenericTokenizer* verwendet). Die zweite SQL-Anweisung verwendet die generische Funktion *Match*, um einen N-Gram-Tokenizer der Größe 3 zur Berechnung des Jaccard-Maßes zu verwenden:

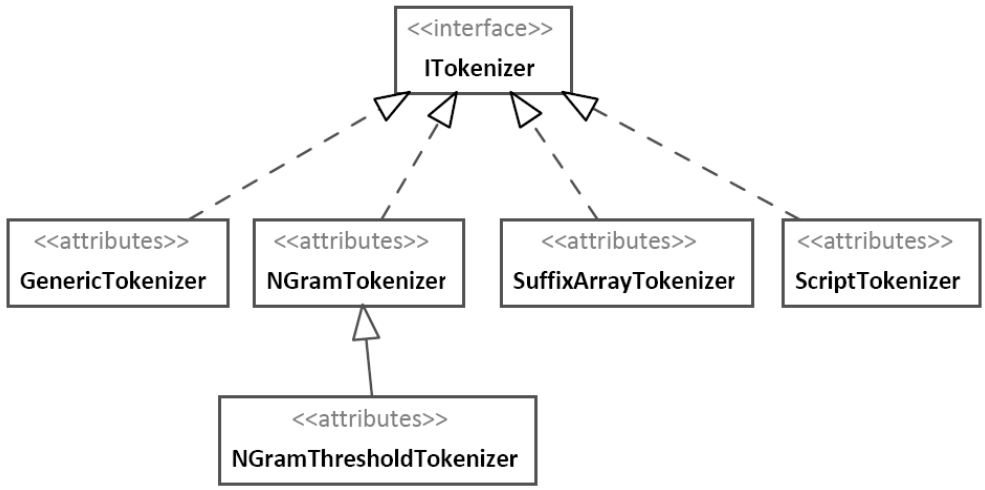
```
SELECT q.Text.MatchingTokenJaccard( 'Hans Muster Hamburg',
                                    'Hamburg, Hans Moster' )

SELECT q.Text.Match( 'Hans Muster Hamburg', 'Hamburg, Hans Moster',
         'FreD.Text.Matching.JaccardApproximateMatching',
         'Core',
         '{Tokenizer=FreD.Text.Tokenizer.NGramTokenizer, NGramSize=3}' )
```

6.2.4 Tokenizer

Algorithmen, die Zeichenketten in bestimmte Segmente (z.B. in Wörter) aufteilen, bezeichnet man als Tokenizer. Neben der Klasse *GenericTokenizer*, über die eine Zeichenkette nach bestimmten Trennzeichen wie z.B. Komma, Semikolon, Punkt usw. geteilt wird, sind in der Klassenbibliothek vier weitere Tokenizer implementiert.

Die Klasse *NGramTokenizer* teilt die Zeichenkette in gleich große Einheiten auf. Die Klassen *SuffixArrayTokenizer* und *NGramThresholdTokenizer* werden vor allem zur Reduktion des Suchraums bei der Duplikaterkennung verwendet. Über die Klasse *ScriptTokenizer* können schließlich eigene Tokenizer über ein Skript zur Verfügung gestellt werden.

Abb. 45 FreD.Text.Tokenizer



Quelle: eigene Darstellung

Im Gegensatz zu den bisherigen Funktionen werden Tokenizer hauptsächlich von anderen Klassen, wie den Algorithmen zur Berechnung Token-basierter Distanz- oder Ähnlichkeitsmaße, als Parameter verwendet. Als Schnittstelle zum RDBMS existieren die beiden generischen Funktionen *Tokenize* und *TokenizeFrequency*. Der Funktion Tokenize wird neben der Klassenbezeichnung, der Bibliothek und benannten Parametern eine Zeichenkette übergeben, die entsprechend dem Algorithmus in Token aufgeteilt wird. Das folgende SQL-Beispiel verwendet die Klasse *SuffixArrayTokenizer* zum Trennen:

```
SELECT q.Text.Tokenize( LastName, ' | ',
             'FreD.Text.Tokenizer.SuffixArrayTokenizer',
             'Core', 'MinSuffixLength=5' )
FROM   AdventureWorks.Person.Contact
```

Die Funktion *TokenizeFrequency* liefert eine Häufigkeitstabelle aller Token für eine Spalte in einer Tabelle zurück. Die Häufigkeitstabelle enthält drei Spalten: Die Spalte „Token“ enthält den aufgetrennten Wert, „Frequency“ gibt die absolute und „FrequencyRelative“ die relative Häufigkeit an. Das folgende Beispiel liefert eine Häufigkeitstabelle von N-Grams der Größe 2 für die Spalte „LastName“ der Tabelle „Person.Contact“.

```
SELECT *
FROM q.Text.TokenizeFrequency(
       'AdventureWorks.Person.Contact', 'LastName', null,
       'FreD.Text.Tokenizer.NGramTokenizer', 'Core', 'NGramSize=2', 0 )
ORDER BY Frequency DESC
```

6.2.5 Erweiterungen

Um eigene Encoder-, Matching-, oder Tokenizer-Verfahren in das Datenqualitäts-Framework zu integrieren, müssen die entsprechenden Schnittstellen oder die virtuellen Methoden der dazugehörigen Basisklasse implementiert werden. So muss eine neue Encoder-Klasse die *IEncoder* oder eine davon abgeleitete Schnittstelle implementieren. Das folgende Beispiel zeigt einen einfachen Encoder in Java, der eine Zeichenkette in Großbuchstaben umwandelt.

```
package DataQuality.Extension;

import java.lang.String.*;

import FreD.Text.Encoder.*;

public class UpperCase implements ITextPreparer
{
      public String Encode( String text )
      {
            if (text == null )
                  return null;

            return text.toUpperCase();
      }
}
```

Zum Registrieren dieser Erweiterung beim Framework existiert die SQL-Prozedur *AddPlugin*. Dieser Funktion wird der Pfad auf die Quellcodedatei oder der Quellcode selbst übergeben sowie der Name der Bibliothek, unter der dieser im RDBMS registriert werden

soll, und eventuell zu verwendende weitere Klassenbibliotheken, auf die im Quellcode referenziert wird. Die folgende SQL-Anweisung registriert den obigen Encoder unter dem Bibliotheksnamen „SampleJava“. Die Klassenbibliothek des Datenqualitäts-Frameworks befindet sich in der Datei „core.dll“ und muss referenziert werden, da in ihr die Schnittstellen beschrieben sind.

```
EXEC q.Tools.AddPlugin 'c:\FreD\samples\Encoder.java',
                       'SampleJava',
                       'c:\FreD\core.dll'
```

Zum Aufruf des Encoders wird die in Abschnitt 6.2.2 vorgestellte generische Encoder-Funktion wie folgt verwendet:

```
SELECT q.Text.Encode( LastName,
            'DataQuality.Extension.UpperCase', 'SampleJava', null )
FROM Person.Contact
```

Benannte Parameter, die an die generische Funktion übergeben werden können, werden als Properties implementiert. Um einen zusätzlichen Property *IgnoreVowels* zum Herausfiltern von Vokalen in einer Zeichenkette zu implementieren, wird die Java-Klasse wie folgt geändert:

```
public class UpperCase implements ITextPreparer
{
      public String Encode(String text)
      {
            if (text == null)
                  return null;

            text = text.toUpperCase();

            if (_ignoreVowels)
            {
                  text = text.Replace("A", "");
                  text = text.Replace("E", "");
                  text = text.Replace("I", "");
                  text = text.Replace("O", "");
                  text = text.Replace("U", "");
            }
            return text;
      }

      /** @property */
      public void set_IgnoreVowels(boolean value)
      {
            _ignoreVowels = value;
      }

      private boolean _ignoreVowels = false;
}
```

Das Setzen des Properties in einer SQL-Anweisung erfolgt im generischen Aufruf über den Parameter *namedParameters* wie folgt:

```
SELECT q.Text.Encode( LastName,
             'DataQuality.Extension.UpperCase', 'SampleJava',
             'IgnoreVowels=true' )
FROM AdventureWorks.Person.Contact
```

Müssen Encoder die *IEncoder*-Schnittstelle implementieren, so ist dies für die Matching-Algorithmen die Schnittstelle *IMatching* oder eine davon abgeleitete Schnittstelle. Der folgende in C# implementierte Matching-Algorithmus überprüft die Übereinstimmung der ersten n Zeichen zweier Zeichenketten. Stimmen diese überein, so wird das Ähnlichkeitsmaß als Quotient übereinstimmender Zeichen und der überprüften Anzahl an Zeichen „n“ errechnet.

```
namespace DataQuality.Extension
{
      public class SimpleMatching : FreD.Text.Matching.ITextMatching
      {
            public int StartLength
            {
                  set
                  {
                        _startLength = value;
                  }
            }

            public double Match( string text1, string text2 )
            {
                  int i = 0;

                  if( string.IsNullOrEmpty( text1 ) ||
                      string.IsNullOrEmpty( text2 ) )
                        return 0.0;

                  text1 = text1.ToUpper();
                  text2 = text2.ToUpper();

                  for( ; i<text1.Length &&
                         i<text2.Length &&
                         i<_startLength; i++ )
                  {
                        char c1 = text1[ i ];
                        char c2 = text2[ i ];

                        if( c1 != c2 )
                              break;
                  }

                  return ( double )i / _startLength;
            }

            private int _startLength = 5;
```

```
      }
}
```

Nachdem das Plug-in über die SQL-Prozedur *AddPlugin* unter dem Bibliotheksnamen „SampleC#“ registriert wurde, kann es beispielhaft wie folgt über die generische Match-Funktion verwendet werden:

```
  SELECT q.Text.Match( 'Muster', 'Musder',
           'DataQuality.Extension.SimpleMatching', 'SampleC#', null )
```

Tokenizer müssen die *ITokenizer* oder eine davon abgeleitete Schnittstelle implementieren. Das folgende Beispiel in Visual Basic implementiert einen Tokenizer, der zum Auftrennen der einzelnen Token das Semikolon als Trennzeichen verwendet:

```
Namespace DataQuality.Extension

      Public Class SimpleTokenizer
            Implements FreD.Text.Tokenizer.ITokenizer

            Public Function Tokenize(ByVal text As String) As String() _
                  Implements FreD.Text.Tokenizer.ITokenizer.Tokenize

                  Dim chars() As Char = {";"}
                  Return text.Split(chars)

            End Function

      End Class

 End Namespace
```

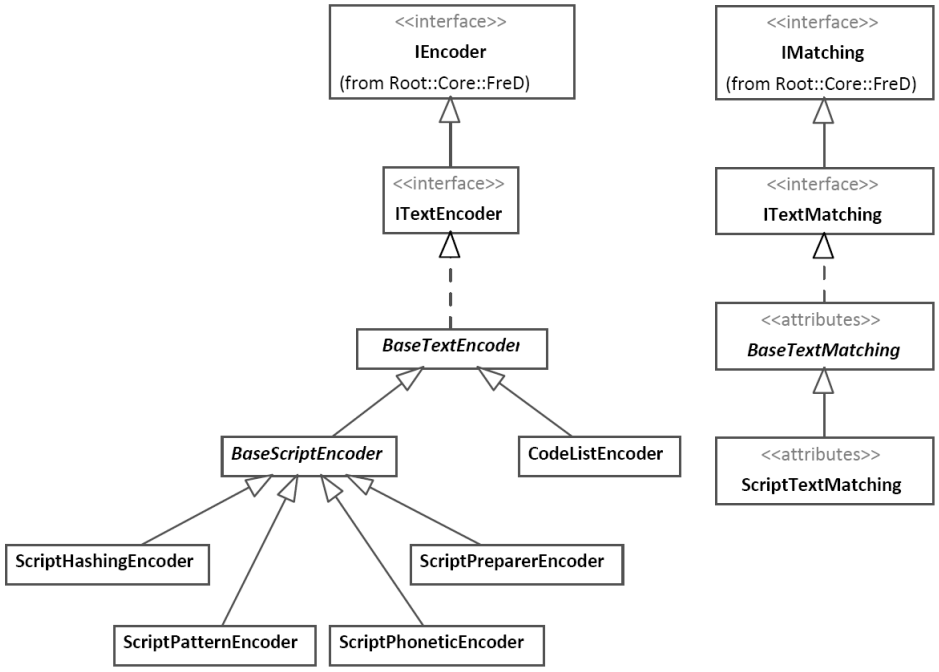
Nachdem die Klasse über *AddPlugin* als Plug-in registriert ist, kann sie über die generische *Tokenize*-Funktion benutzt werden.

Neben dem bisher beschriebenen Verfahren zum Erweitern des Frameworks, gibt es spezielle Klassen, die die Implementierung eines speziellen Plug-ins erleichtern. Alle mit dem Namen *Script* beginnenden Klassen sind über eine spezielle *AddPlugin*-Prozedur zur Verfügung gestellt, die bei der Implementierung das Deklarieren der Schnittstelle usw. abnimmt. So kann damit die Implementierung eines Encoders etwas kürzer ausfallen. Ein Encoder, der die Eingabe in Großbuchstaben umwandelt, wird mit der speziellen Plug-in-Prozedur wie folgt registriert und aufgerufen:

```
EXEC Text.AddPluginPreparerScript
         'return text.ToUpper();', 'Plugin.Preparer'

SELECT LastName,
       q.Text.Encode( LastName,
       'Quality.Text.Encoder.Encoder', 'Plugin.Preparer', null )
FROM AdventureWorks.Person.Contact
```

Erweiterungen über ein eigenes Plug-in sind zwar flexibel, aber auch aufwändig. Alle im Framework implementierten Verfahren und Algorithmen nutzen deshalb ausgiebig benannte Parameter, um das Verhalten der jeweiligen Klasse möglichst flexibel zu steuern. Z.B. kann den Token-basierten Verfahren zum Vergleichen zweier Zeichenketten ein anderer als der *GenericTokenizer* zugewiesen werden. So können beispielsweise über den *NGramTokenizer* auch typographische Fehler berücksichtigt werden. Ein Zeichen-basierter Algorithmus wiederum kann auch über einen benannten Parameter einen Tokenizer verwenden, um dadurch einen hybriden Algorithmus, wie in Abschnitt 3.2.2.3 beschrieben, zu verwenden. Ein spezieller Meta-Encoder wird über die Klasse *CodeListEncoder* implementiert, die es erlaubt, beliebige Encoder zu einem gemeinsamen Encoder zu kombinieren (siehe Abb. 46).

Abb. 46 Erweiterungen der Basis



Quelle: eigene Darstellung

6.3 Verstehen

6.3.1 Allgemein

Verfahren zum Verstehen und Analysieren der Daten finden sich im Schema „Understand“. Neben den Eigenschaften einer Spalte, gehören hierzu die Analyse von Fremd-/Primärschlüsselbeziehungen und das Suchen nach Geschäftsregeln in den Daten über Regelinduktionsverfahren.

Bei den Eigenschaften einer Spalte kann man unterscheiden zwischen Daten- und Schemaeigenschaften. Dateneigenschaften ergeben sich aus den Daten selbst, wie z.B. statistische Kennzahlen, Anzahl NULL-Werte usw. Schemaeigenschaften dagegen betreffen die Struktur einer Spalte, wie z.B. Datentyp usw.

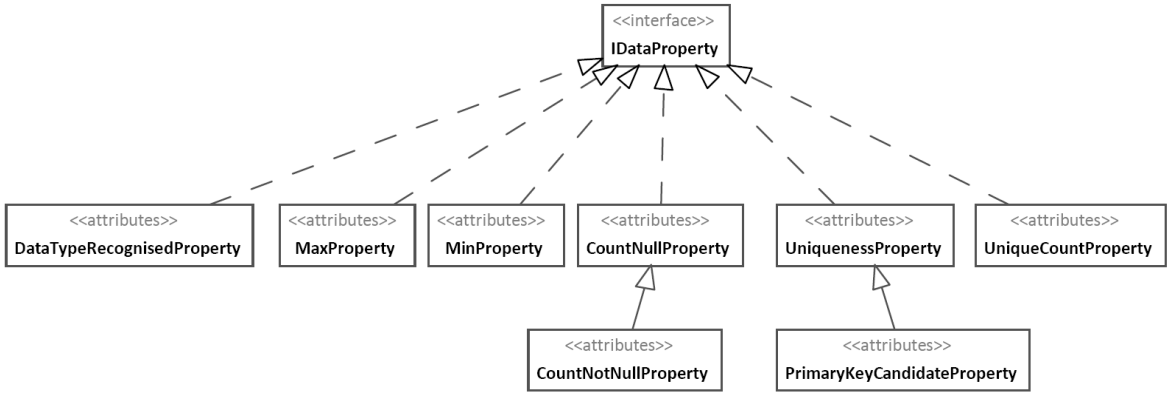
Die Analyse von Primärschüsseln und Fremdschlüsseln betrifft sowohl die Suche nach einem geeigneten Primärschlüssel in einer Tabelle als auch die Suche der Beziehung eines Fremdschlüssels in einer Tabelle zu einem Primärschlüssel in einer anderen Tabelle.

Im Folgenden werden zunächst die SQL-Routinen zum Ermitteln von Daten- und Schemaeigenschaften beschrieben, es folgt die Analyse von Primär- und Fremdschlüsseln und Verfahren zur Regelinduktion. Im letzten Abschnitt wird erläutert, wie das Framework um eigene Verfahren erweitert werden kann.

6.3.2 Daten- und Schemaeigenschaften

Dateneigenschaften werden aus den Daten einer Spalte selbst ermittelt. Dabei gibt es sowohl generelle als auch spezielle Dateneigenschaften, die vom Datentyp abhängen. Alle Klassen, die eine Dateneigenschaft implementieren, werden von der Schnittstelle *IDataProperty* abgeleitet.

Abb. 47 DataProperty (allgemein)



Quelle: eigene Darstellung

Abb. 47 zeigt alle in der Klassenbibliothek implementierten Methoden zur Ermittlung allgemeiner vom Datentyp unabhängigen Dateneigenschaften. Die Klasse *UniqueCountProperty* ermittelt die Anzahl eindeutiger Werte. *UniquenessProperty* zeigt den Anteil eindeutiger Werte im Verhältnis zur Gesamtanzahl und *PrimaryKeyCandidate* gibt abhängig von einem Schwellwert an, ob es sich um einen Primärschlüssel handelt. Über *CountNullProperty* und *CountNotNullProperty* wird die Anzahl von NULL bzw. vorhandener Werte ermittelt. *MinProperty* und *MaxProperty* suchen die minimalen und maximalen Werte für eine Spalte, wobei deren Anzahl über einen benannten Parameter gesetzt werden kann. *DataTypeRecognisedProperty* schließlich gibt für eine Spalte den erkannten Datentyp zurück. Hierüber können Unterschiede in dem deklarierten Datentyp der Spalte und den wirklichen gespeicherten Daten überprüft werden.

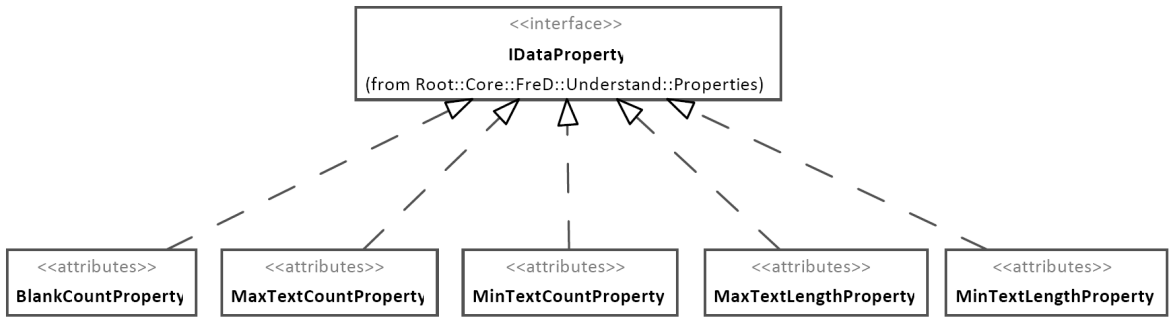
Dateneigenschaften werden im RDBMS als benutzerdefinierte Aggregatfunktionen implementiert. Neben den spezifischen existiert eine generische Aggregatfunktion zum Aufruf einer beliebigen Dateneigenschaftsklasse bzw. eines Plug-ins. Das folgende Beispiel zeigt zunächst eine Anweisung, bei der die Dateneigenschaften über spezifische Aggregatfunktionen aufgerufen werden, sowie eine weitere SQL-Anweisung mit einem generischen Aufruf zur Ermittlung der NULL-Werte:

```
SELECT q.Understand.PropertiesDataTypeRecognised( AddressLine2 ),
       q.Understand.PropertiesCountNull( AddressLine2 ),
       q.Understand.PropertiesCountNotNull( AddressLine2 ),
       q.Understand.PropertiesMax( AddressLine2 ),
       q.Understand.PropertiesMin( AddressLine2 )
FROM   Person.Address
```

```
SELECT q.Understand.PropertiesDataProperty(
          '{FreD.Understand.Properties.MinProperty|Core|MinCount=10}'+
          COALESCE( AddressLine2, '{NULL}') )
FROM   Person.Address
```

Spezielle Klassen, die Dateneigenschaften für Spalten mit dem Datentyp Text implementieren, sind in der folgenden Abbildung dargestellt. Über die Klasse *BlankCountProperty* wird die Anzahl an Werten ermittelt, die nur aus Leerzeichen bestehen. *MaxTextCountProperty* und *MinTextCountProperty* geben die Anzahl der Werte mit dem kürzesten bzw. längsten Text zurück und *MinTextLengthProperty* und *MaxTextLengthProperty* die jeweils kürzeste und längste Länge.

Abb. 48 DataProperty (Text)



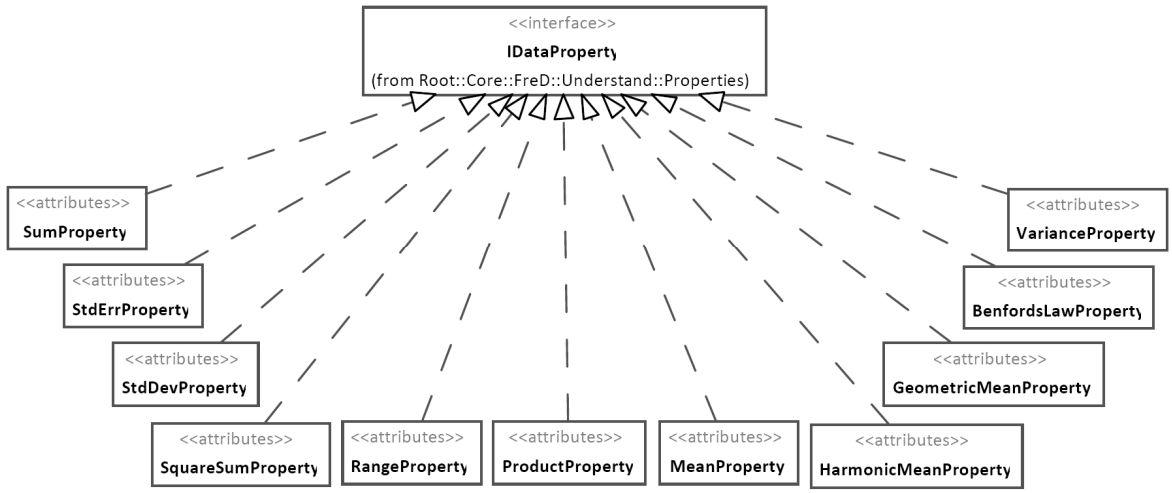
Quelle: eigene Darstellung

Folgende SQL-Anweisung verwendet die spezifischen Funktionen zum Ermitteln der Dateigenschaften für Text:

```
SELECT q.Understand.PropertiesMinTextLength( PostalCode ),
       q.Understand.PropertiesMinTextCount( PostalCode ),
       q.Understand.PropertiesMaxTextLength( PostalCode ),
       q.Understand.PropertiesMaxTextCount( PostalCode ),
       q.Understand.PropertiesBlankCount( PostalCode )
FROM   Person.Address
```

Spezielle Klassen für Zahlen berechnen vor allem statistische Kennzahlen wie Mittelwert, Standardabweichung usw. Bei Zahlen wird zusätzlich unterschieden zwischen Dateneigenschaften für Zahlen allgemein und für Fließkommazahlen. Abb. 49 zeigt alle Dateneigenschaften für Zahlen. Neben den Klassen für bekannte statistische Kennzahlen (*VarianceProperty*, *StdErrProperty*, *StdDevProperty*, *SquareSumProperty*, *MeanProperty* usw.) existiert die Klasse *BenfordsLawProperty*, die eine Überprüfung der Verteilung der einzelnen Ziffern nach Benford vornimmt.

Abb. 49 DataProperty (Zahlen)



Quelle: eigene Darstellung

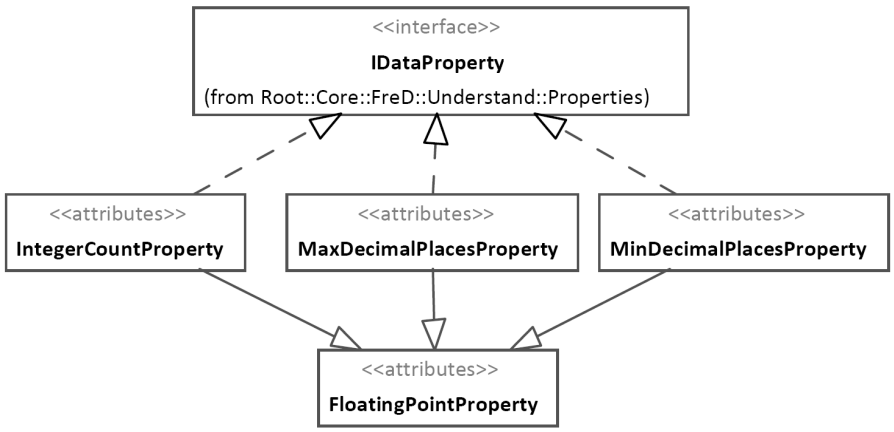
Die folgende erste SQL-Anweisung liefert für eine Spalte alle im Framework implementierten Dateneigenschaften über die jeweiligen spezifischen SQL-Aggregatfunktionen zurück. Die zweite SQL-Anweisung errechnet den Mittelwert über die generische Aggregatfunktion, wobei die maximalen 10 und minimalen 10 Werte nicht berücksichtigt werden (benannter Parameter *TrimmedCount*):

```
SELECT q.Understand.PropertiesBenfordsLaw( ListPrice ),
       q.Understand.PropertiesGeometricMean( ListPrice ),
       q.Understand.PropertiesHarmonicMean( ListPrice ),
       q.Understand.PropertiesMean( ListPrice ),
       q.Understand.PropertiesProduct( ListPrice ),
       q.Understand.PropertiesRange( ListPrice ),
       q.Understand.PropertiesSquareSum( ListPrice ),
       q.Understand.PropertiesStdDev( ListPrice ),
       q.Understand.PropertiesStdErr( ListPrice ),
       q.Understand.PropertiesSum( ListPrice ),
       q.Understand.PropertiesVariance( ListPrice )
FROM   Production.Product

SELECT q.Understand.PropertiesMean( ListPrice ),
       q.Understand.PropertiesDataProperty(
 '{FreD.Understand.Properties.Number.MeanProperty|Core|TrimmedCount=10}'+
 COALESCE( CAST( ListPrice AS NVARCHAR(50) ), '{NULL}' ) )
FROM Production.Product
```

Neben diesen für alle Zahlen zu ermittelnden Dateneigenschaften, existieren drei Implementierungen ausschließlich für Fließkommazahlen. Über die Klasse *IntegerCountProperty* wird die Anzahl an Ganzzahlen ermittelt. *MaxDecimalPlacesProperty* und

MinDecimalPlacesProperty liefern die maximale bzw. minimale Anzahl an Nachkommastellen.

Abb. 50 DataProperty (Fließkommazahlen)



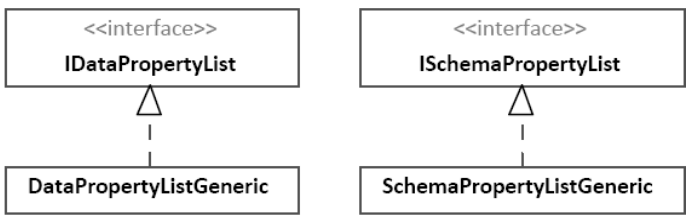
Quelle: eigene Darstellung

Folgende SQL-Anweisung liefert die Dateneigenschaften für Fließkommzahlen über deren spezifische Funktionen zurück:

```
SELECT q.Understand.PropertiesIntegerCount( ListPrice ),
       q.Understand.PropertiesMaxDecimalPlaces( ListPrice ),
       q.Understand.PropertiesMinDecimalPlaces( ListPrice )
FROM   Production.Product
```

Neben den Dateneigenschaften, die über Aggregationsfunktionen in SQL abgebildet werden, existiert eine weitere SQL-Funktion, die als Rückgabewert eine Tabelle zurückliefert. Diese Tabelle enthält alle in einer Bibliothek vorhandenen Dateneigenschaften und deren Werte für eine oder mehrere Spalten einer Tabelle. So wie es diese Funktion für Dateneigenschaften gibt, existiert eine weitere SQL-Funktion für Schemaeigenschaften.

Abb. 51 Daten- und Schemaeigenschaften



Quelle: eigene Darstellung

Klassen, die eine Tabelle mit Daten- oder Schemaeigenschaften zurückliefern, müssen die Schnittstellen *IDataPropertyList* bzw. *ISchemaPropertyList* implementieren. Die Klassenbibliothek enthält für Dateneigenschaften eine Implementierung in der Klasse *DataPropertyListGeneric* und für Schemaeigenschaften in *SchemaPropertyListGeneric*. *DataPropertyListGeneric* ermittelt in einer übergebenen Bibliothek alle Klassen, die die Schnittstelle *IDataProperty* implementieren und ermittelt für eine oder mehrere Spalten die entsprechenden Dateneigenschaften. *SchemaPropertyListGeneric* gibt alle Schemaeigenschaften für ein oder mehrere Spalten einer Tabelle über ADO.NET zurück.

Die folgende SQL-Anweisung ermittelt für die übergebenen Spalten „Color“ und „ListPrice“ alle Dateneigenschaften, die in der Bibliothek „core“ über die Schnittstelle *IDataProperty* implementiert sind.

```
SELECT *
FROM q.Understand.PropertyListDataGeneric(
                          'AdventureWorks.Production.Product',
                          'Color, ListPrice',
                          null, 'core' )
```

Daneben existiert noch die generische SQL-Funktion *PropertyListData*, über die eigene Implementierungen als Plug-in aufgerufen werden können. Entsprechend gibt es für Schemaeigenschaften eine SQL-Funktion *PropertyListSchema* zum generischen Aufrufen eigener Implementierungen. Das folgende SQL-Beispiel zeigt noch einmal die Verwendung der Klasse *SchemaPropertyListGeneric* über die SQL-Funktion *PropertyListSchemaGeneric*, die in diesem Beispiel für alle Spalten einer Tabelle alle ermittelten Schemaeigenschaften zurückliefert.

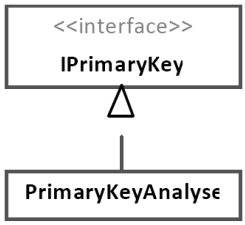
```
SELECT *
FROM q.Understand.PropertyListSchemaGeneric(
                     'AdventureWorks.Person.Address', '*' )
```

6.3.3 Primär- und Fremdschlüssel

Klassen, die Algorithmen zur Erkennung von Primärschlüsseln in einer Tabelle einbinden, müssen die Schnittstelle *IPrimaryKey* mit der Methode *Analyse* implementieren. In der Klassenbibliothek des Frameworks ist zurzeit eine einfache Methode in der Klasse

PrimaryKeyAnalyse implementiert, die aus einer Liste von übergebenen Spalten zunächst die ausfiltert, die aufgrund des Datentyps als Primärschlüssel nicht in Frage kommen (z.B. Fließkommazahlen). Für jede der übriggebliebenen Spalten wird die Anzahl eindeutiger Werte ermittelt. Liegt diese Anzahl über einem vorgegebenen Schwellwert, so wird diese Spalte als Primärschlüsselkandidat in einer Liste zurückgegeben. Dieses Verfahren wird für jeweils eine einzelne Spalte und für Kombinationen aus zwei Spalten durchgeführt.

Abb. 52 PrimaryKey



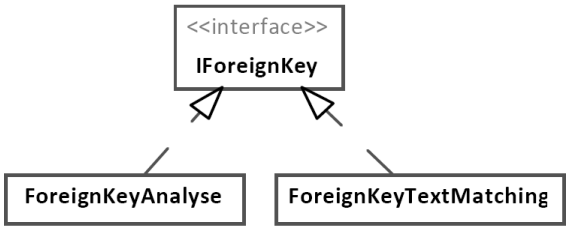
Quelle: eigene Darstellung

Zur Nutzung im RDBMS existiert die spezifische SQL-Funktion *PrimaryKeyAnalyse* und die generische Funktion *PrimaryKey*. Folgendes SQL-Beispiel zeigt zunächst den spezifischen Aufruf am Beispiel der Tabelle „Person.Contact“ und danach den generischen Aufruf.

```
SELECT * FROM q.Understand.PrimaryKeyAnalyse(
             'AdventureWorks.Person.Contact', '*', null );

SELECT * FROM q.Understand.PrimaryKey(
             'AdventureWorks.Person.Contact', '*', null,
             'FreD.Understand.Dependencies.PrimaryKey',
             'Core', 'Threshold=0.5' );
```

Zur Erkennung von Primär-/Fremdschlüsselbeziehungen zwischen zwei Tabellen muss eine Klasse die Schnittstelle *IForeignKey* umsetzen. In der Klassenbibliothek existieren hierzu zwei Implementierungen. Die Klasse *ForeignKeyAnalyse* überprüft die Übereinstimmung zwischen zwei Spalten instanzbasiert anhand der Dateninhalte. Die Klasse *ForeignKeyTextMatching* ist schemabasiert und verwendet einen approximativen Textvergleichsalgorithmus, um ähnliche Spaltenbezeichnungen zu erkennen.

Abb. 53 ForeignKey



Quelle: eigene Darstellung

Die folgenden SQL-Anweisungen ermitteln Primär-/Fremdschlüsselbeziehungen über die spezifischen SQL-Funktionen zu diesen beiden Klassen.

```
SELECT * FROM q.Understand.ForeignKeyAnalyse(
              'AdventureWorks.Sales.SalesOrderHeader',
              'CustomerId, SalesPersonId, TerritoryId, BillToAddressId',
              'CustomerId < 100',
              'AdventureWorks.Sales.Customer',
              'CustomerId, TerritoryId', 'CustomerId < 100' )

SELECT * FROM q.Understand.ForeignKeyTextMatchingAnalyse(
              'AdventureWorks.Sales.SalesOrderHeader', '*', null,
              'AdventureWorks.Sales.Customer', '*', null )
```

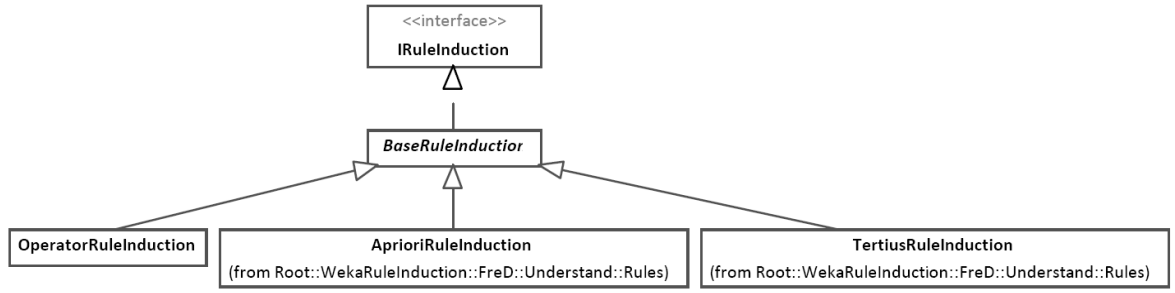
6.3.4 Regelinduktion

Zur Implementierung von Regelinduktions-Klassen muss die Schnittstelle *IRuleInduction* realisiert werden. Die in der Klassenbibliothek implementierte Klasse *OperatorRuleInduction* erkennt zwischen zwei Spalten, welcher Anteil an zwei Werten eines Datensatzes größer, kleiner oder gleich ist.

Über das Plug-in „WekaRuleInduction“ wurden die an der Universität von Waikato entwickelten Regelinduktionsverfahren „Tertius“ und „Apriori“ eingebunden.[472]

[472] Siehe hierzu das in Java entwickelte Open Source Projekt „Weka“ („Waikato Environment for Knowledge Analysis“): http://www.cs.waikato.ac.nz/ml/weka/

Abb. 54 RuleInduction



Quelle: eigene Darstellung

Zum Suchen von Regeln in einem Datenbestand existieren die generische SQL-Funktion *RuleInduction* und die spezifische *RuleInductionOperator*.

Die erste der folgenden drei SQL-Anweisungen erzeugt Regeln über die Funktion *RuleInductionOperator*, um für eine Tabelle Bestell-, Auslieferungs- und Fälligkeitsdaten zu überprüfen. Die zwei darauf folgenden Anweisungen verwenden die Klassen *AprioriRuleInduction* und *TertiusRuleInduction* über einen generischen SQL-Aufruf, um Regeln in einer Produkttabelle zu finden.

```
SELECT "Rule", Support,  Frequency
FROM q.Understand.RuleInductionOperator(
     'AdventureWorks.Sales.SalesOrderHeader',
     'OrderDate, DueDate, ShipDate', null )
```

```
SELECT *
FROM q.Understand.RuleInduction(
     'AdventureWorks.Production.Product',
     'Color,ProductLine, Class', null,
     'FreD.Understand.Rules.AprioriRuleInduction', 'WekaRuleInduction',
     'MinItemFrequency=3, Delta=0.05')
```

```
SELECT "Rule", Indicator
FROM q.Understand.RuleInduction(
     'AdventureWorks.Production.Product',
     'Color,ProductLine, Class', null,
     'FreD.Understand.Rules.TertiusRuleInduction', 'WekaRuleInduction',
     'AutoPreProcess=false')
```

6.3.5 Erweiterungen

Zur Erweiterung der Funktionalitäten müssen Plug-ins Klassen implementieren, die die entsprechenden Schnittstellen für Daten-/Schemaeigenschaften, Primär-/Fremdschlüsselerkennung und Regelinduktion umsetzen. Um z.B. das Framework um eine eigene Dateneigenschaft zu erweitern, muss eine Klasse die Schnittstelle *IDataPropery* implementieren. Diese Schnittstelle besteht aus den drei Methoden *Process*, *Terminate*, *GetValue* und den Eigenschaften *Name* und *Description*. Da eine Dateneigenschaft ein aggregierter Wert ist, erhält die Klasse über die Methode *Process* die einzelnen Werte. *Terminate* beendet die Übergabe der Daten und über *GetValue* kann der aggregierte Wert ermittelt werden. Das folgende in C# geschriebene Plug-in ermittelt für eine Spalte mit Textwerten die Anzahl derer, die ausschließlich in Großbuchstaben gespeichert sind.

```
namespace myDataProperties
{
	[System.Serializable]
	public class UpperTextCount:FreD.Understand.Properties.IDataProperty
	{
		public string Name
		{
			get { return "Text.UpperTextCount"; }
		}

		public string Description
		{
			get { return "Count of text only in upper case."; }
		}

		public object Process( object value )
		{
			if( value != null )
			{
				string text = value.ToString();

				if( text == text.ToUpper() )
					_count++;
			}

			return _count;
		}
```

```
            public object Terminate()
            {
                  return _count;
            }

            public object GetValue()
            {
                  return _count;
            }

            private int _count = 0;
      }
}
```

Wie in Abschnitt 6.2.5 zu Erweiterungen der Basis beschrieben wird die Quellcodedatei über die SQL-Prozedur *AddPlugin* beim Framework registriert. Über die generische SQL-Aggregationsfunktion *PropertiesDataProperty* oder über die SQL-Funktion *PropertyListDataGeneric* kann die neue Dateneigenschaft für eine Tabellenspalte ermittelt werden. Das folgende Beispiel registriert zunächst das Plug-in und verwendet dann die beiden SQL-Funktionen, um die neue Eigenschaft für ausgewählte Spalten zu berechnen.

```
EXEC q.Tools.AddPlugin 'c:\FreD\samples\DataProperty.cs',
                        'PluginDataProperty','c:\FreD\bin\core.dll'

SELECT q.Understand.PropertiesDataProperty(
         '{myDataProperties.UpperTextCount|PluginDataProperty}'+
         COALESCE(Class, '{NULL}') )
FROM  Production.Product

SELECT *
FROM q.Understand.PropertyListDataGeneric(
         'AdventureWorks.Production.Product',
         'Color, Weight', null, 'PluginDataProperty' )
```

Um Dateneigenschaften aus mehreren Bibliotheken für ausgewählte Spalten zu ermitteln, können der Funktion *PropertyListDataGeneric* mehrere registrierte Bibliotheksnamen übergeben werden. Das folgende Beispiel gibt neben der Dateneigenschaft der Bibliothek „PluginDataProperty“ auch alle Dateneigenschaften der Bibliothek „core“ aus.

```
SELECT *
FROM   q.Understand.PropertyListDataGeneric(
         'AdventureWorks.Production.Product', '*', null,
         'PluginDataProperty|Core' )
ORDER BY TypeName
```

6.4 Verbessern

6.4.1 Allgemein

Verfahren zum Verbessern und Korrigieren von Daten werden im Schema „Improve“ gespeichert. Hierzu gehört neben der Duplikaterkennung, dem Standardisieren, Korrigieren und Anreichern von Daten die Verwaltung von Regeln. Über Regeln kann vor allem domänenspezifisch die Geschäftslogik im RDBMS berücksichtigt werden. Die Nutzung von Regeln im RDBMS hat den Vorteil, dass auf der Datenbank aufsetzende Anwendungssysteme bei Änderung oder Hinzufügen von Regeln nicht oder nur geringfügig geändert werden müssen.

Im Folgenden wird die Regelverwaltung erläutert. Neben dem Klassendiagramm gehören hierzu vor allem der Aufbau und die Syntax zur Beschreibung von Regeln. Danach folgt die Beschreibung zum Standardisieren, Korrigieren und Anreichern von Daten und zur Duplikaterkennung.

6.4.2 Regelverwaltung

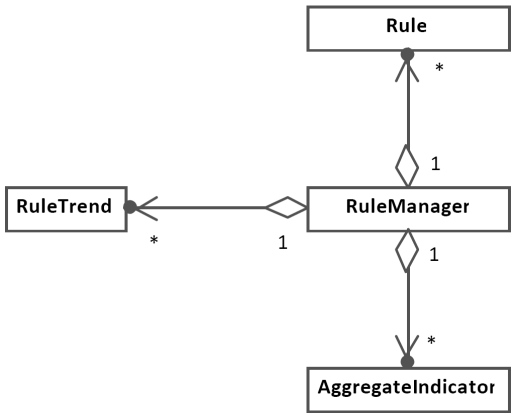
Die Verwaltung von Regeln erfolgt über die Klasse ***RuleManager***. Über diese Klasse werden Regeln neu angelegt, geändert, gelöscht und auf ihre Syntax hin überprüft. Der Regelmanager verwaltet die einzelnen Regeln über eine Liste von Instanzen der Klasse ***Rule***.

Die Klasse ***Rule*** enthält alle Informationen einer Regel. Hierzu gehören allgemeine Information wie Kategorie, Erstellungs-/Änderungsdatum, Kurzbeschreibung und Status (inaktiv, aktiv). Die Regel selbst besteht aus dem Regelereignis, über das fünf verschiedene Ereignisse unterschieden werden: OnInsert, OnUpdate, OnChange, OnDelete, OnTimer und OnApplication. Diese Ereignisse beziehen sich auf eine Tabelle, deren Namen bei der Erstellung einer Regel angegeben wird. Beim Anlegen einer Regel wird dann für die ersten vier Ereignisse ein Trigger verwendet, um auf diese Ereignisse zu reagieren. OnTimer-Ereignisse dagegen beziehen sich auf zeitliche Ereignisse. Hierfür existiert die Dienstanwendung „EventDetector“, die bei Eintreten des Zeitereignisses den Regelmanager

benachrichtigt. Neben dem Ereignis werden die Bedingung und die Aktion festgelegt, die bei Eintreten der Bedingung ausgeführt wird.

Neben den Regeln selbst verwaltet die Klasse *RuleManager* den Verlauf von Qualitätsindikatoren für jede Regel. Für jede Regel mit einer Bedingung kann ein Indikator berechnet werden, der überprüft, bei wie vielen Datensätzen die Bedingung korrekt bzw. inkorrekt ist. Der Qualitätsindikator wird als Quotient aus korrekten und Summe der korrekten und inkorrekten Datensätze errechnet. Um den Trend einer Regel zu ermitteln, werden die Veränderungen der Indikatoren über Instanzen der Klasse *RuleTrend* gespeichert. Über die Klasse *AggregateIndicator* können mehrere Regeln zu einem gemeinsamen Indikator zusammengefasst werden.

Abb. 55 Regelverwaltung



Quelle: eigene Darstellung

SQL-Routinen zur Regelverwaltung befinden sich im Schema „Rules“. Zum Anlegen, Löschen und Ändern existieren die Prozeduren *InsertRule*, *UpdateRule*, *DeleteRule*. Die Funktion *Rules* zeigt alle vorhandenen Regeln in Form einer Tabelle an. Zur Definition einer Regel können die gesamten vom RDBMS unterstützten SQL-Sprachelemente verwendet werden.

Das folgende Beispiel zeigt das Anlegen einer Regel mit einer Bedingung, die das Eingabeformat der Spalte „Phone“ überprüft. Ereignis und Aktion sind nicht definiert.

```
EXEC q.Rules.InsertRule
          'R1',
          null,
          'q.Text.PatternsSimple( Phone ) = ''999*999*9999''',
          null,
          'Format',
          'active',
          10,
          'AdventureWorks.Person.Contact'
```

Folgende Parameter werden der Prozedur *InsertRule* übergeben: Kurzbezeichnung der Regel („R1“), Ereignis, Bedingung, Aktion, Kategorie, Status, Gewicht und Name der Tabelle, auf die sich die Regel bezieht.

Die Prozeduren *Valid* und *Invalid* liefern für diese Regel eine Liste mit allen gültigen oder ungültigen Datensätzen zurück. Beiden Prozeduren wird das Kürzel der Regel übergeben und optional eine SQL-Anweisung, die auf die gültige bzw. ungültige Regel angewendet wird. Die folgende SQL-Anweisung gibt für die angelegte Regel „R1“ alle gültigen Datensätze zurück.

```
EXEC q.Rules.Valid 'R1', null
```

Um die Anzahl aller ungültigen Datensätze zu erhalten, wird die entsprechende SQL-Anweisung als Parameter übergeben.

```
EXEC q.Rules.Invalid 'R1',
          'SELECT COUNT(*) FROM AdventureWorks.Person.Contact'
```

Um eine Regel durch eine Anwendung selbst auszulösen, kann die Prozedur *ExecRule* verwendet werden. Jede Regel kann dabei unabhängig vom definierten Ereignis immer applikationsbezogen ausgeführt werden.[473] Das folgende Beispiel führt die angelegte Regel aus und erzeugt eine Ausnahme, wenn die Bedingung für einen Datensatz nicht erfüllt ist.

```
EXEC q.Rules.ExecRule 'R1'
```

Soll ein Datensatz bei Verletzung einer Bedingung generell nicht geändert bzw. neu eingefügt werden, so muss neben der Bedingung das entsprechende Ereignis OnInsert, OnUpdate oder

[473] Für eine Regel, die ausschließlich von einer Anwendung ausgelöst wird, sollte jedoch als Ereignis OnApplication verwendet werden

OnChange[474] gesetzt sein. Bei dem nächsten Beispiel wird jede Änderung an einem Datensatz zurückgewiesen, wenn die Spalte „Size“ den NULL-Wert enthält:

```
EXEC q.Rules.InsertRule
         'R2',
         'OnChange',
         'Size IS NOT NULL',
         null,
         'Vollständigkeit',
         'active',
         80,
         'AdventureWorks.Production.Product'
```

Über das Anlegen einer Aktion kann das Standardverhalten des Auslösens einer Ausnahme selbst festgelegt werden. Zum einen kann eine Regelverletzung einfach nur protokolliert werden. Zum anderen können die einzufügenden Daten geändert werden. Die folgende SQL-Anweisung legt eine neue Regel an, die das Geburtsdatum überprüft. Ist die Bedingung nicht erfüllt, wird das Problem protokolliert:

```
EXEC q.Rules.InsertRule
       'R3',
       'OnChange',
       'YEAR(GETDATE()) - YEAR(BirthDate) NOT BETWEEN 21 AND 65 ',
       'EXEC q.Tools.Log ''file://c:\Log.txt'',''Konsist.'', ''Alter!''',
       'Wertebereich',
       'active',
       100,
       'AdventureWorks.HumanResources.Employees'
```

Über die Aktion einer Regel können die Daten auch automatisch z.B. über einen Predictor (siehe Abschnitt 6.4.3) angepasst werden. Die folgende Regel ermittelt über einen Predictor das Geschlecht aufgrund des Vornamens. Wurde die Bedingung der Regel verletzt, werden die Daten entsprechend geändert.

```
EXEC q.Rules.InsertRule
     'R4',
     'OnInsert',
     'Gender NOT IN (''W'', ''M'' ) OR Gender IS NULL',
     'UPDATE q.temp.inserted SET Gender =
            CAST( q.Improve.PredictorPredictLookup( ''Geschlecht'',
            q.Tools.Param( FirstName, 1 ) ) AS CHAR(1))',
     'Wertebereich',
     'active',
     20,
     'AdventureWorks.HumanResources.Employees'
```

[474] OnChange entspricht den beiden Ereignissen OnInsert und OnUpdate

Sollen ungültige Daten einer Regel einmalig korrigiert werden, so kann dies über den zweiten Parameter der bereits erwähnten Prozedur *Invalid* erfolgen. Das folgende Beispiel korrigiert unter Verwendung eines Predictors das Geschlecht abhängig vom Vornamen über die Regel „R4“.

```
EXEC q.Rules.Invalid
     'R4',
     'UPDATE AdventureWorks.HumanResources.Employees
          SET Gender =
          CAST( q.Improve.PredictorPredictLookup( ''Geschlecht'',
          q.Tools.Param( FirstName, 1 ) ) AS CHAR(1))'
```

Eine weitere Möglichkeit besteht darin, die Daten nicht unmittelbar, sondern zu einem bestimmten Zeitpunkt zu korrigieren. Dazu wird die Regel „R4“ so geändert, dass sie auf ein Zeitereignis reagiert und die Änderungen in der Originaltabelle vornimmt:

```
EXEC q.Rules.InsertRule
     'R4',
     'OnTimer at 23:30 every day',
     'Gender NOT IN (''W'', ''M'' ) OR Gender IS NULL',
     'UPDATE AdventureWorks.HumanResources.Employees
          SET Gender = CAST( q.Improve.PredictorPredictLookup(
          ''Geschlecht'', q.Tools.Param( FirstName, 1 ) ) AS CHAR(1))',
     'Wertebereich',
     'active',
     80,
     'AdventureWorks.HumanResources.Employees'
```

Die Bedingungen der bisherigen Regeln beziehen sich immer auf einzelne Instanzen und nicht auf aggregierte Eigenschaften mehrerer Daten. Um z.B. zu überprüfen, ob es nicht mehr als zwei unterschiedliche Geschlechtsbezeichnungen gibt oder der Anteil einer Produktkategorie einen bestimmten Prozentsatz überschreitet, können sich Regeln auf mehrere Datensätze beziehen. Über das spezielle Schlüsselwort „BOOL{“ kann daher ein sogenannter Tabellenausdruck als Bedingung definiert werden. Die erste der beiden folgenden SQL-Anweisungen erstellt eine Regel, die überprüft, ob mehr als zwei Geschlechtsbezeichnungen verwendet werden und die zweite, ob der Anteil an männlichen Mitarbeitern größer als 50% ist.

```
EXEC q.Rules.InsertRule
     'R5',
     null,
     'BOOL{q.Tools.AggregateValue(
          ''AdventureWorks.HumanResources.Employees'',
          ''COUNT( DISTINCT Gender )'', null )} > 2',
     'q.Tools.SendMail
          ''mail.server.de'',
          ''cordts@server.de'',
          ''cordts@server.de'',
```

```
          ''DQ (AdventureWorks.HumanResources.Employees)'',
          ''More than 2 gender flags''; PRINT ''Mail versendet!''',
     'Wertebereich',
     'active',
     10,
     'AdventureWorks.HumanResources.Employees'


EXEC q.Rules.InsertRule
     'R6',
     null,
     'BOOL{q.Tools.RelativeFrequency(
          ''AdventureWorks.HumanResources.Employees'',
          ''Gender'',
          ''''''M''''''')} > CAST( 0.50 AS REAL )',
     null,
     'Konsistenz',
     'active',
     0,
     'AdventureWorks.HumanResources.Employees'
```

Für jede Regel wird die Anzahl der Werte ermittelt, die die Regel erfüllen bzw. nicht erfüllen. Aus diesen beiden Zahlen wird dann der Datenqualitätsindikator nach folgender Formel errechnet:

$$Indicator(n_{valid}, n_{invalid}) = \frac{n_{valid}}{n_{valid} + n_{invalid}}$$

mit

n_{valid} Anzahl der Werte, die Regel erfüllen

$n_{invalid}$ Anzahl der Werte, die Regel nicht erfüllen

Die Berechnung der Indikatoren für jede Regel muss durch die SQL-Prozedur *UpdateStatistics* initiiert werden. *UpdateStatistics* berechnet den aktuellen Indikator und speichert ihn zweimal im Repository: Einmal zu jeder Regel und zusätzlich in der Tabelle „RulesTrend“ mit einem Zeitstempel, um die Historie und damit den Trend eines Indikators zu verfolgen. Über die bereits oben erwähnte SQL-Funktion *Rules* kann die Tabelle mit allen Regeln und deren aktuellen Indikatoren ausgegeben werden. Über die Funktion *RulesTrend* kann der zeitliche Verlauf der Indikatoren für alle Regeln als Tabelle ausgegeben werden.

Das folgende Beispiel aktualisiert zunächst die Indikatoren aller Regeln. Danach werden alle Regeln mit Ihren aktuellen Indikatoren und schließlich die Historie der Indikatoren ausgegeben.

```
EXEC q.Rules.UpdateStatistics

SELECT Code, Invalid, Valid, Indicator FROM q.Rules.Rules()
SELECT Code, Date, Indicator FROM q.Rules.RulesTrend()
```

Um die Indikatoren automatisch z.B. einmal wöchentlich zu aktualisieren, kann eine neue Regel erstellt werden, die als Aktion die Prozedur *UpdateStatistics* aufruft. Das folgende Beispiel errechnet die Indikatoren zu Beginn jeder Woche neu.

```
EXEC q.Rules.InsertRule
     'Statistik1',
     'OnTimer at 00:30 every monday',
     null,
     'EXEC q.Rules.UpdateStatistics',
     'Statistik',
     'active',
     0,
     null
```

Zum Aggregieren mehrerer Indikatoren zu einer Datenqualitätskennzahl, gibt es die SQL-Funktion *AggregateIndicator*. Darüber kann der aktuelle Indikator für mehrere Regeln oder auch für einen bestimmten Zeitraum errechnet werden. Der erste Parameter der Funktion ist eine SQL-Bedingung, mit der die Regelmenge eingeschränkt wird. Der zweite und dritte Parameter dient zum Begrenzen des Zeitraums. Wird für alle Parameter kein Wert übergeben, so wird über alle Regeln ein Indikator auf Grundlage der aktuellen Indikatoren errechnet.

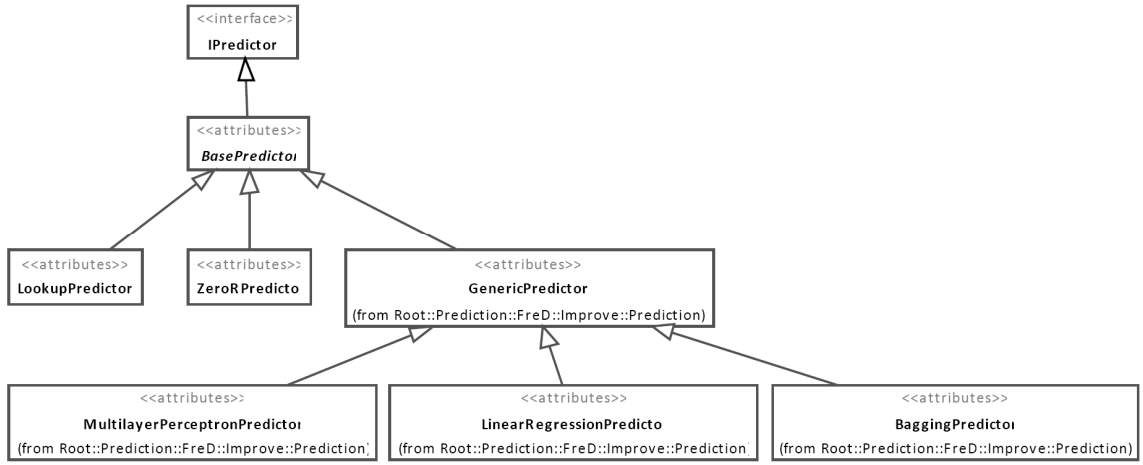
Die folgenden SQL-Anweisungen geben zunächst einen Indikator über alle Regeln aus. Dann wird der Gesamtindikator für alle Regeln der Kategorie „Vollständigkeit“ ausgegeben. Die dritte Anweisung ermittelt für alle Regeln den aggregierten Indikator über einen bestimmten Zeitraum und die vierte für eine bestimmte Datenbank.

```
SELECT * FROM q.Rules.AggregateIndicator(
    null, null, null )

SELECT * FROM q.Rules.AggregateIndicator(
    'Rules.Category = ''Vollständigkeit''', null, null )

SELECT * FROM q.Rules.AggregateIndicator(
    null, '01.01.2008', '30.06.2009' )

SELECT * FROM q.Rules.AggregateIndicator(
    'Rules."Table" LIKE ''AdventureWorks.%''', null, null )
```

6.4.3 Daten standardisieren, korrigieren, anreichern

Ein Predictor ist im Datenqualitäts-Framework eine Methode, die auf Basis von Eingabewerten einen Ausgabewert ermittelt. Predictoren können Algorithmen des maschinellen Lernens sein oder aber auch Methoden, die Werte aus einer einfachen Nachschlagetabelle ermitteln. Predictoren müssen die Schnittstelle *IPredictor* implementieren, über die die Eingabe- und Ausgabeattribute festgelegt werden und eine Methode *Build* den entsprechenden Algorithmus umsetzt. In der Klassenbibliothek implementieren die beiden Klassen *LookupPredictor* und *ZeroRPredictor* diese Schnittstelle. Über die Klasse *ZeroRPredictor* wird auf Grundlage des einfachen maschinellen Lernverfahrens 0R ein Wert klassifiziert.[475] Über die Klasse *LookupPredictor* wird ein Wert vorhergesagt, indem über die Eingabewerte in einer Nachschlagetabelle der Ausgabewert ermittelt wird.

Abb. 56 Predictor



Quelle: eigene Darstellung

Neben diesen beiden Verfahren sind über das Plug-in „WekaPrediction“ zahlreiche Algorithmen der an der Universität von Waikato in Java implementierten Klassenbibliothek „Weka“ eingebunden. Exemplarisch hierfür sind in Abb. 56 die Klassen *MultiLayerPerceptronPredictor* für die Klassifikation über ein neuronales Netz,

[475] Bei 0R wird für ein numerisches vorherzusagendes Attribut der Mittelwert , ansonsten der Modus vorhergesagt.

LinearRegressionPredictor zur linearen Regression und *BaggingPredictor* als Meta-Klassifizierer abgebildet.[476]

Um einen Predictor in SQL für bestimmte Daten zu generieren bzw. lernen zu lassen, existiert im Schema „Improve“ die Prozedur *PredictorBuild* und zur Vorhersage die Funktion *PredictorPredict*. Ein einmal gelernter Predictor wird im Repository der Datenqualitäts-komponente persistent gespeichert und kann über die SQL-Prozedur *PredictorDelete* wieder gelöscht werden. Informationen über einen vorhandenen Predictor erhält man über die Funktion *PredictorInfo*.

Das folgende Beispiel erzeugt zunächst einen Predictor als ID3-Entscheidungsbaum über die generische Prozedur *PredictorBuild*, um die Spalte „City“ auf Basis der Spalten „StateProvinceId“ und „PostalCode“ vorherzusagen.

```
EXEC q.Improve.PredictorBuild
  'AdventureWorks.Person.Address',
  'CAST( StateProvinceId AS NVARCHAR(10) ) AS StateProvinceId,
  PostalCode',
  'City', null, 'PredictorCity', 'FreD.Improve.Prediction.Id3Predictor',
  'WekaPrediction',
  null

SELECT DISTINCT
    City,
    q.Improve.PredictorPredict( 'PredictorCity',
          q.Tools.Param( CAST( StateProvinceId AS NVARCHAR(10) ), 0 ) +
          q.Tools.Param( PostalCode, 1 ) ) AS CityPredicted
FROM  AdventureWorks.Person.Address
WHERE LTRIM( City ) <>
      LTRIM( CAST( q.Improve.PredictorPredict( 'PredictorCity',
          q.Tools.Param( CAST( StateProvinceId AS NVARCHAR(10) ), 0 ) +
          q.Tools.Param( PostalCode, 1 ) ) AS NVARCHAR(50)) )
```

Das zweite Beispiel verwendet die spezifische Prozedur *PredictorBuildLookup*, um einen Predictor zu erzeugen, der aus der Nachschlagetabelle „q.Lookup.[City.US]“ über die Spalte „PostalCode“ den Wert für „City“ ermittelt. Über die zweite SQL-Anweisung werden alle Orte ausgegeben, deren Namen nicht mit dem ermittelten übereinstimmen.

[476] Siehe hierzu das in Java entwickelte Open Source Projekt „Weka“ („Waikato Environment for Knowledge Analysis“): http://www.cs.waikato.ac.nz/ml/weka/

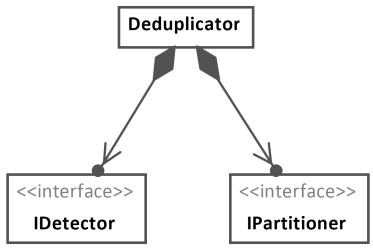
```
EXEC q.Improve.PredictorBuildLookup
     'q.Lookup."City.US"',
     'ZIP', 'City',  null, 'PredictoryCity'


SELECT DISTINCT PostalCode, City,
     q.Improve.PredictorPredict( 'PredictoryCity',
          q.Tools.Param( PostalCode, 1 ) ) AS CityPredicted
FROM Person.Address a, Person.StateProvince b
WHERE
     a.StateProvinceID = b.StateProvinceID AND
     b.CountryRegionCode = 'US' AND
     LTRIM(City) <> LTRIM(CAST( q.Improve.PredictorPredict(
          'PredictoryCity',
          q.Tools.Param( PostalCode, 1 ) ) AS NVARCHAR(250) ) )
```

6.4.4 Duplikaterkennung und -fusion

Die Erkennung von Duplikaten erfolgt über die Klasse Deduplicator, die jeweils eine Instanz einer Detector- und einer Partitioner-Klasse instanziiert. Über den Partitioner wird die nach Duplikaten zu untersuchende Tabelle in Gruppen von Datensätzen partitioniert, innerhalb derer Duplikate wahrscheinlich sind. Aus diesen Partitionen liefert der Partitioner dem Deduplicator jeweils immer zwei Datensätze, für die der Detector überprüft, inwieweit sie auf dasselbe reale Objekt verweisen.

Partitioner-Klassen müssen die Schnittstelle *IPartitioner* und Detector-Klassen die Schnittstelle *IDetector* implementieren.

Abb. 57 Deduplicator

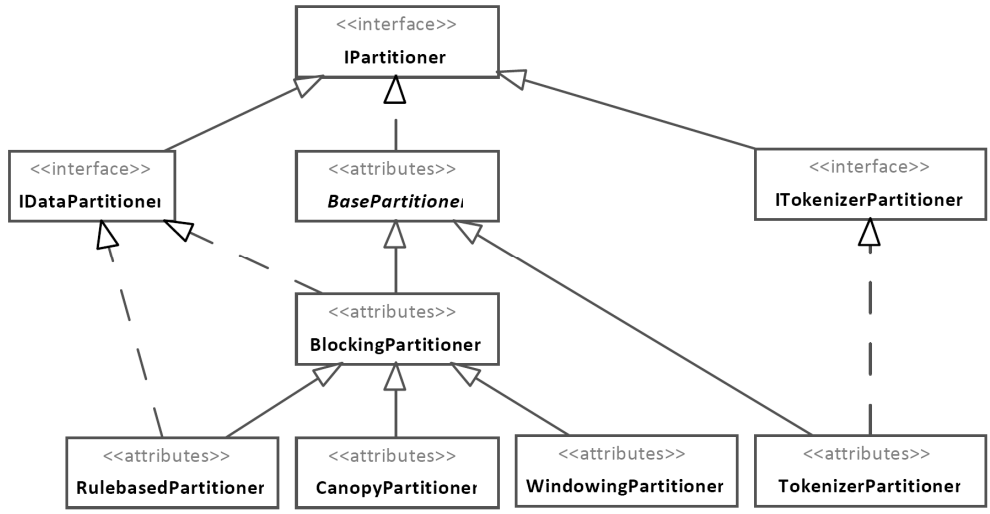


Quelle: eigene Darstellung

Zum Partitionieren der Datensätze einer Tabelle sind in der Klassenbibliothek fünf unterschiedliche Algorithmen implementiert. Über den *BlockingPartitioner* wird das in Abschnitt 3.5.3.2 beschriebene Blocking, über den *WindowingPartitioner* die Sorted Neighborhood

Methode bzw. das Windowing, über den *CanopyPartitioner* das Canopy Clustering und über den *TokenizerPartitioner* u.a. das Suffix Array und das N-Gram basierte Blocking implementiert. Über die weitere Klasse *RuleBasedPartitioner* kann eine Gruppierung über SQL-Funktionen erfolgen. Zum Erkennen und späteren Gruppieren von Duplikaten werden der Originaltabelle drei zusätzliche Spalten hinzugefügt: Eine Spalte „q.ID“ zur eindeutigen Kennung eines Datensatzes, „q.MatchKey“[477] zur Speicherung des generierten Sortierschlüssels und „q.Group“ zum Zusammenfassen von Duplikaten.

Abb. 58 Partitioner



Quelle: eigene Darstellung

Partitionierer, die sich auf eine bestimmte Tabelle und bestimmte Spalten beziehen, werden über die generische SQL-Prozedur *DeduplicationBuildPartioner* definiert und unter einem Namen im Repository der Datenqualitätskomponente gespeichert. Über die SQL-Funktion *DeduplicationInfoPartitioner* können Informationen zu einem gespeicherten Partitioner wieder abgerufen und über *DeduplicationDeletePartitioner* kann dieser gelöscht werden.

Die folgenden ersten beiden SQL-Anweisungen erzeugen jeweils einen *BlockingPartitioner* und einen *WindowingPartitioner*, die den Metaphone-Code bzw. sortierte Zeichen der Spalte „LastName“ und „FirstName“ zum Sortieren verwenden. Die darauf folgenden drei SQL-Anweisungen erzeugen *TokenizerPartitioner*. Der erste erzeugt N-Grams der Größe 3, der zweite auch und berücksichtigt zusätzlich einen Schwellwert von 0,8 und der dritte erzeugt

[477] Ein Partitionierer mit mehr als einem Sortierschlüssel (*TokenizerPartitioner*) speichert die Sortierschlüssel in einer zusätzlichen Tabelle im Repository der Datenqualitätskomponente.

Suffixe bis zu einer Länge von 3 Zeichen. Die vorletzte Anweisung erstellt einen *CanopyPartitioner*, der alle Werte der zum Gruppieren angegebenen Spalten über den Sift3-Algorithmus als Standardwert miteinander vergleicht, um Gruppen zu bilden. Als Schwellwert beim approximativen Textvergleich des *CanopyPartitioners* wird standardmäßig eine Übereinstimmung von 0,5 erwartet. Die letzte Anweisung schließlich erzeugt einen regelbasierten Partitionierer, indem Regeln zum Vergleich von Spalten festgelegt werden.

```
EXEC q.Improve.DeduplicationBuildPartitioner
 'AdventureWorks.Person.Contacts',
 'q.Text.PatternsCharSort(LastName+FirstName)',
 'FirstName, LastName, Phone',
 null,
 'BlockingContact',
 'FreD.Improve.Deduplication.BlockingPartitioner',
 'Core', null

EXEC q.Improve.DeduplicationBuildPartitioner
 'AdventureWorks.Person.Contacts',
 'q.Text.PhoneticDoubleMetaphone(LastName) +
  q.Text.PhoneticDoubleMetaphone(FirstName)',
 'FirstName, LastName, Phone',
 null,
 'WindowingContact',
 'FreD.Improve.Deduplication.WindowingPartitioner',
 'Core', null

EXEC q.Improve.DeduplicationBuildPartitioner
 'AdventureWorks.Person.Contacts',
 'q.Text.PatternsCharSort(LastName)',
 'FirstName, LastName, Phone',
 null,
 'TokenizerNGramContact',
 'FreD.Improve.Deduplication.TokenizerPartitioner',
 'Core', null

EXEC q.Improve.DeduplicationBuildPartitioner
 'AdventureWorks.Person.Contacts',
 'q.Text.PatternsCharSort(LastName)',
 'FirstName, LastName, Phone',
 null, 'TokenizerNGramThresholdContact',
 'FreD.Improve.Deduplication.TokenizerPartitioner',
 'Core',
 '{Tokenizer=FreD.Text.Tokenizer.NGramThresholdTokenizer}'

EXEC q.Improve.DeduplicationBuildPartitioner
 'AdventureWorks.Person.Contacts',
 'q.Text.PatternsCharSort(LastName)',
 'FirstName, LastName, Phone',
 null,
 'TokenizerSuffixArrayContact',
 'FreD.Improve.Deduplication.TokenizerPartitioner',
 'Core',
 '{Tokenizer=FreD.Text.Tokenizer.SuffixArrayTokenizer}'
```

```
EXEC q.Improve.DeduplicationBuildPartitioner
 'AdventureWorks.Person.Contacts',
 'q.Text.PatternsCharSort(LastName)',
 'FirstName, LastName, Phone',
 null,
 'CanopyContact',
 'FreD.Improve.Deduplication.CanopyPartitioner',
 'Core', null

EXEC q.Improve.DeduplicationBuildPartitioner
 'AdventureWorks.Person.Contacts',
 'q.Text.MatchingSift3( a.LastName, b.LastName) > 0.5',
 'FirstName, LastName, Phone', null,
 'RulebasedContact',
 'FreD.Improve.Deduplication.RulebasedPartitioner',
 'Core', null
```

Nachdem das Verfahren zum Partitionieren der Datensätze durchgeführt ist, muss ein Detector festgelegt werden. Dieser vergleicht jeweils zwei Datensätze, um festzustellen, ob diese dasselbe reale Objekt bezeichnen. Eine Klasse, die einen Detector implementiert, muss die Schnittstelle *IDetector* realisieren. In der Klassenbibliothek sind fünf verschiedene Verfahren zur Erkennung implementiert. Zum einen sind die probabilistischen Verfahren nach Fellegi & Sunter und der TF-IDF-Algorithmus implementiert. Daneben existiert ein Verfahren, das Matchkeys auf Basis des Partitionierers bildet und ein deterministischer Detector, der alle Werte untereinander vergleicht und die maximale Übereinstimmung für jede Spalte summiert. Danach bildet der Quotient aus Summe und Anzahl der Spalten den Grad der Übereinstimmung zwischen den beiden Datensätzen. Beim TF-IDF-Algorithmus sind zwei Verfahren in den Klassen *TFIDFDetector* und *SoftTFIDFDetector* implementiert. Die Klasse *SoftTFIDFDetector*-Klasse entspricht einer Implementierung von W. Cohen in Java. Hierbei werden die Gewichte nach folgender Formel berechnet[478]:

$$w_{i,j} = \log(tf_{i,j} + 1) \cdot \log \frac{N}{n_i}$$

Im Gegensatz dazu erlaubt die *TFIDFDetector*-Klasse es, über benannte Parameter die Formel zur Berechnung der Gewichte selbst festzulegen.

Um zu entscheiden, ob es sich bei zwei Datensätzen um Duplikate handelt, muss an einen Detector ein Schwellwert übergeben werden.[479] Für die probabilistischen Verfahren ist es

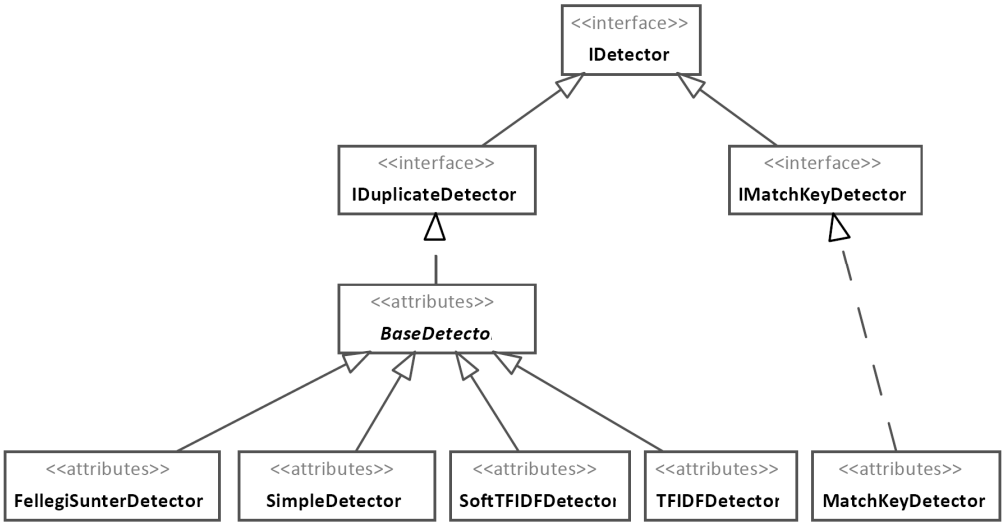
[478] Cohen, W. u.a.: A Comparison of String Distance Metrics for Name-Matching Tasks; America Assocation for Artificial Intelligence; Menlo Park; 2003; S. 2

[479] Der Standardwert für den Schwellwert beträgt 0,8

sinnvoll, eine Häufigkeitstabelle als benannten Parameter zu übergeben, um daraus entsprechend die Gewichte für einen Wert zu berechnen.[480] Zum automatischen Erstellen einer Häufigkeitstabelle existiert die Prozedur *BuildFrequencyTable* im Schema „Tools“.

Genau wie beim Partitionierer auch wird der Detector im Repository der Datenqualitätskomponente gespeichert. Dies erfolgt analog zum Partitionierer über die SQL-Prozedur *DeduplicationBuildDetector*. Entsprechend gibt es eine Funktion *DeduplicationInfoDetector*, um Informationen über einen Detector abzurufen und die Prozedur *DeduplicationDeleteDetector* um diesen zu löschen.

Abb. 59 Detector



Quelle: eigene Darstellung

Das folgende Beispiel erzeugt zunächst die Häufigkeitstabelle „PersonContact“ für die übergebenen Spalten der Tabelle „Person.Contacts“. Danach wird für jedes der implementierten Verfahren ein Detector angelegt. Beim *MatchKeyDetector* wird über benannte Parameter der Token-basierte Vergleichsalgorithmus auf den Dice-Algorithmus gesetzt (Standard ist Jaccard) und das Tracing eingeschaltet (Ausgabe der Datensatzpaare und deren Gewichte während der Erkennung). Für die Klassen *TFIDFDetector*, *SoftTFIDFDetector* und *FellegiSunterDetector* wird als Häufigkeitstabelle „PersonContact“ als Parameter angegeben.

[480] Ohne Übergabe einer Häufigkeitstabelle wird bei TF-IDF für jeden zu vergleichenden Wert ein Gewicht von 1,0 verwendet; bei Fellegi & Sunter wird als Standard für die Wahrscheinlichkeit Matches ein Wert von 0,95 und bei Unmatches von 0,01 verwendet

Zusätzlich wird beim *TFIDFDetector* noch eine eigene Formel zur Berechnung der Gewichte gesetzt und beim *FellegiSunterDetector* ein anderer Schwellwert.

```
EXEC q.Improve.DeduplicationBuildDetector
    'DetectorSimple',
    'FreD.Improve.Deduplication.SimpleDetector',
    'Core',
    null

EXEC q.Improve.DeduplicationBuildDetector
    'DetectorMatchKey',
    'FreD.Improve.Deduplication.MatchKeyDetector',
    'Core',
    '{TokenComparator=FreD.Text.Matching.DiceApproximateMatching},
    Trace=true'

EXEC q.Improve.DeduplicationBuildDetector
    'DetectorTFIDF',
    'FreD.Improve.Deduplication.TFIDFDetector',
    'Core',
    'FrequencyTable=PersonContact, WeightFormula=tf * log(idf)'

EXEC q.Improve.DeduplicationBuildDetector
    'DetectorSoftTFIDF',
    'FreD.Improve.Deduplication.SoftTFIDFDetector',
    'Core',
    'FrequencyTable=PersonContact'

EXEC q.Improve.DeduplicationBuildDetector
    'DetectorFellegiSunter',
    'FreD.Improve.Deduplication.FellegiSunterDetector',
    'Core',
    'Threshold=10.0, FrequencyTable=PersonContact'
```

Zum Starten der Duplikaterkennung existieren die beiden SQL-Prozeduren *Deduplicate* und *DeduplicateIncremental* im RDBMS. *Deduplicate* dient zum einmaligen Erkennen von Duplikaten in einem Datenbestand. Der Prozedur *DeduplicateIncremental* wird zusätzlich der Name einer Tabelle übergeben, die den gleichen Aufbau wie die im Partitioner angegebene Tabelle hat und für deren Datensätze die Duplikaterkennung durchgeführt wird. Über diese Prozedur kann vor allem innerhalb einer Regel, die beim Einfügen neuer Datensätze ausgeführt wird, auf Duplikate geprüft werden.

Das folgende Beispiel verwendet die Prozedur *Deduplicate*, um über den angelegten Partitioner „BlockingContact“ und den Detector „DetectorTFIDF“ Duplikate in der Tabelle „Person.Contacts“ zu ermitteln. Zuvor wird über die Prozedur *BuildFrequencyTable* eine Häufigkeitstabelle für die Werte aller übergebenen Spalten erstellt, die dann bei der Duplikaterkennung zur Berechnung der Gewichte verwendet wird.

```
EXEC q.Tools.BuildFrequencyTable
    'PersonContacts',
    'AdventureWorks.Person.Contacts',
    'Title, FirstName, LastName, Phone', null, 1, 0


EXEC q.Improve.Deduplicate 'BlockingContact', 'DetectorTFIDF', null
```

Das zweite Beispiel erzeugt eine Regel, die beim Einfügen neuer Datensätze in die Tabelle „Person.Contacts“ aufgerufen wird. Diese Regel überprüft, ob bei den neuen Datensätzen Duplikate vorhanden sind, um gegebenenfalls eine Ausnahme auszulösen.

```
EXEC q.Rules.InsertRule
    'R1Insert',
    'OnInsert',
    null,
    'EXEC q.Improve.DeduplicateIncremental
          ''q.temp.inserted'',
          ''BlockingContact'',
          ''DetectorTFIDF'', null;
     IF EXISTS (SELECT [q.Group]
                     FROM q.temp.inserted
                     WHERE [q.Group] IS NOT NULL )
          RAISERROR( ''Person with same data already exist!'', 18, 1 );',
    'Konsistenz',
    'active',
    0,
    'AdventureWorks.Person.Contacts'
```

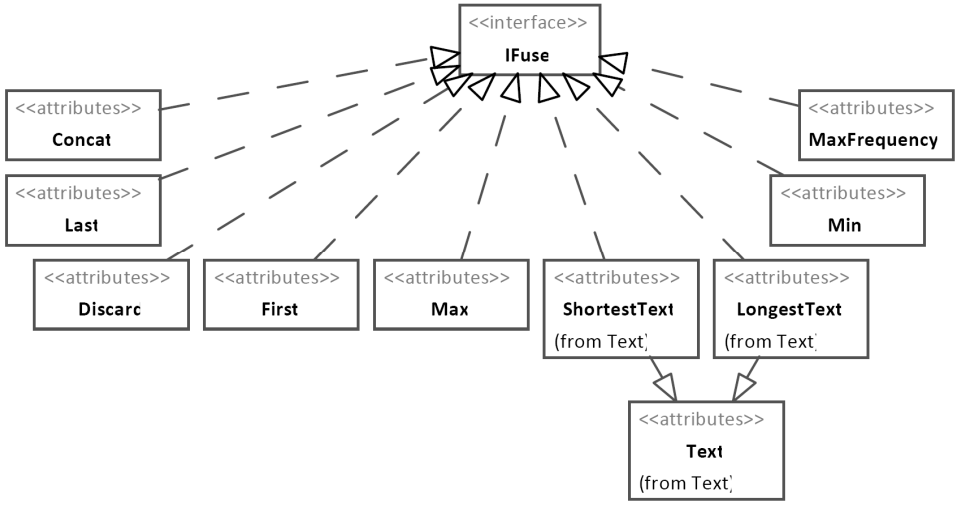
Nachdem alle Duplikate erkannt und entsprechend in der Spalte „q.Group“ markiert wurden, können über eine gruppierte SQL-Anweisung alle zusammengehörenden Datensätze zusammengefasst werden. Wie in Abschnitt 3.5.3.4 beschrieben, kann dies über eine GROUP BY-Anweisung zusammen mit benutzerdefinierten Aggregationsfunktionen, die zum Zusammenführen von Duplikaten geeignet sind, erfolgen.

In der Klassenbibliothek müssen die als Auflösungsfunktionen bezeichneten Aggregationsfunktionen die Schnittstelle *IFuse* implementieren. Neben den in SQL üblichen Aggregationsfunktionen implementiert die Klassenbibliothek neun weitere Auflösungsfunktionen. Die Klasse *Concat* verbindet die Werte einer Gruppe zu einem gemeinsamen, *Last* verwendet den letzten Wert der Gruppe und *First* den ersten, *Max* und *Min* verwenden jeweils den maximalen bzw. minimalen Wert, *ShortestText* und *LongestText* den kürzesten bzw. längsten Wert, *MaxFrequency* den am häufigsten auftretenden Wert in der Gruppe und *Discard* liefert nur dann kein NULL zurück, wenn alle Werte der Gruppe identisch sind.

Die folgende SQL-Anweisung gruppiert alle in der Tabelle „Person.Contacts“ gefundenen Duplikate.

```
SELECT q.Improve.DeduplicationLongestText( LastName ),
       q.Improve.DeduplicationConcat( FirstName ),
       q.Improve.DeduplicationFirst( Phone )
FROM   Person.Contacts
WHERE  [q.Group] IS NOT NULL
GROUP BY [q.Group]
```

Abb. 60 Fuse



Quelle: eigene Darstellung

6.4.5 Erweiterungen

Um Daten über eigene Verfahren zu standardisieren, anzureichern oder zu korrigieren, muss eine Klasse die Schnittstelle *IPredictor* implementieren und als Plug-in beim Datenqualitäts-Framework registriert werden. Die *IPredictor*-Schnittstelle definiert zwei Methoden. Die Methode *Build* dient zum Erstellen des Predictors und *Predict* zur Vorhersage eines Werte auf Basis des übergebenen Datensatzes. Daneben müssen zwei Properties *FieldClass* und *Fields* definiert werden. *FieldClass* enthält den Namen der vorherzusagenden Spalte und *Fields* die Eingabedaten.

Das folgende Plug-in implementiert diesmal nicht direkt die Schnittstelle, sondern ist von der Basisklasse *BasePredictor* abgeleitet, die diese realisiert. Bei Verwendung der Basisklasse

müssen statt der beiden Methoden *Build* und *Predict* die beiden virtuellen Methoden *DoBuild* und *DoPredict* überschrieben werden, um den Predictor umzusetzen. Der folgende Quellcode erzeugt einen Predictor, der als Eingabewert einen Vornamen erwartet und daraus entweder das Geschlecht oder das Land für diesen Vornamen aus der Tabelle „q.Lookup.Gender“ ausliest.

```
using System;
using System.Data.SqlClient;
using FreD.Data;
using FreD.Data.Providers;
using FreD.Improve.Prediction;

namespace MyPredictor
{
  /// <summary>
  /// siehe hierzu:
  /// J. Michael: Anredebestimmung anhand des Vornamens; erschienen in: c't
  /// 17/07; Heise Verlag; Hannover; 2007; S. 182-183
  /// und http://www.heise.de/ct/ftp/07/17/182/
  /// </summary>

  [Serializable()]
  public class GenderPredictor : BasePredictor
  {
    protected override void DoBuild( string table, string fields,
                                     string fieldClass, string where )
    {
      if( fields.ToLower() != "name" )
          throw new ArgumentException( "Fields must be 'name'" );

      if( fieldClass == null || ( fieldClass.ToLower() != "gender" &&
          fieldClass.ToLower() != "country" ) )
        throw new ArgumentException(
                 "Field class must be 'Gender' or 'Country'" );
    }

    protected override object DoPredict( FreD.Data.Row row )
    {
      using( SqlServerProvider provider = new SqlServerProvider() )
      {
        string name = ( ( string )row.Values[ 0 ] ).Replace( ' ', '+' );

        using( IReader reader = provider.GetTableReader(
               "SELECT * FROM q.Lookup.Gender WHERE Name = @name",
               new SqlParameter[] { new SqlParameter( "@name", name ) } ) )
        {
          row = reader.Read( 0 );

          if( row != null )
          {
            if( FieldClass.ToLower() == "gender" )
              return translateGender( ( ( string )row[ 0 ] ).Trim() );

            return translateCountry( row, reader.Columns );
          }
        }
      }
```

```
    return null;
  }
  …
```

Die Verwendung in SQL sieht dann folgendermaßen aus:

```
EXEC q.Tools.AddPlugin
          'C:\FreD\samples\Predictor.cs', 'SamplePredictor',
          'C:\FreD\Core.dll|System.Data.dll'

EXEC q.Improve.PredictorBuild
     'Gender', 'Name', 'Gender', null,  'Geschlecht',
     'MyPredictor.GenderPredictor', 'SamplePredictor', null

EXEC q.Improve.PredictorBuild
     'Gender', 'Name', 'Country', null, 'Land',
     'MyPredictor.GenderPredictor', 'SamplePredictor', null

SELECT FirstName,
       q.Improve.PredictorPredict( 'Geschlecht', FirstName ),
       q.Improve.PredictorPredict( 'Land', FirstName )
FROM   AdventureWorks.Person.Contact
```

Zur Duplikaterkennung können Plug-ins implementiert werden, indem diese die Schnittstelle *IMatchKeyDetector* bzw. *IDuplicateDetector* für einen Detector und *IPartitioner* für einen Partitionierer implementieren. Zur Erstellung eigener Auflösungsfunktionen zur Fusion mehrerer Datensätze ist die Schnittstelle *IFuse* umzusetzen.

Das folgende Beispiel implementiert einen eigenen Detector, der die Werte eines Datensatzpaares einfach auf exakte Gleichheit überprüft. Dazu implementiert dieser die in der Schnittstelle *IDuplicateDetector* definierte Methode *DuplicateCheck* und die beiden Eigenschaften *GenericComparator* und *Threshold*. Zusätzlich ist die Eigenschaft *Trace* der Schnittstelle *IDetector* umzusetzen.

```
namespace MyDetector
{
  [System.Serializable()]
  public class ExactDetector :
              FreD.Improve.Deduplication.IDuplicateDetector
  {
    public double Threshold
    {
      get { return _threshold; }
      set { _threshold = value; }
    }

    public FreD.IMatching GenericComparator
    {
      get { return null; }
      set { }
    }
```

```
    public bool Trace
    {
      get { return _trace; }
      set { _trace = value; }
    }

    public string DuplicateCheck(
                         object[] values1,
                         object[] values2,
                         FreD.IMatching[] comparators )
    {
      double sum = 0.0;

      for( int i = 0; i < values1.Length; i++ )
      {
        System.IComparable compare = values1[ i ] as System.IComparable;

        // Exakte Übereinstimmung zweier Spaltenwerte überprüfen
        if( compare != null && compare.CompareTo( values2[ i ] ) == 0 )
          sum += 1.0;
      }

      // Summe durch Anzahl Spalten
      double similarity = sum / values1.Length;

      if( similarity >= Threshold )
        return string.Format( "Similarity = {0}, Sum = {1}",
                              similarity, sum );
      return null;
    }

    private double _threshold = 0.8;
    private bool   _trace = false;
  }
}
```

Um den Detector im RDBMS zu verwenden, muss dieser zunächst als Plug-in registriert werden. Danach wird über die Prozedur *DeduplicationBuildDetector* der Detector zur Duplikaterkennung eingerichtet und über *Deduplicate* verwendet.

```
  EXEC q.Tools.AddPlugin
       'C:\FreD\samples\Detector.cs', 'SampleDetector', 'C:\FreD\core.dll'

  EXEC q.Improve.DeduplicationBuildPartitioner
       'AdventureWorks.Person.Contact',
       'q.Text.PatternsCharSort(LastName+SUBSTRING(FirstName, 1, 5))',
       'q.Text.PatternsCharSort(LastName),
        q.Text.PatternsCharSort(SUBSTRING(FirstName, 1, 5 )),
        q.Text.PatternsCharSort(SUBSTRING(Phone, 1, 5)),
        SUBSTRING(EmailAddress, 1, 5)',
        null, 'BlockingContact',
        'FreD.Improve.Deduplication.BlockingPartitioner', 'Core', null

  EXEC q.Improve.DeduplicationBuildDetector
       'DetectorExact', 'MyDetector.ExactDetector', 'SampleDetector',
       'Threshold=0.7, Trace=true'

  EXEC q.Improve.Deduplicate 'BlockingContact', 'DetectorExact', null
```

6.5 Steuern

6.5.1 Allgemein

Sind alle notwendigen Regeln in einer Datenbank erkannt und aktiviert, so sollte deren Einhaltung überprüft und gesteuert werden. Verfahren hierzu befinden sich einerseits im Schema „Control“ und andererseits – wie bereits im Abschnitt 6.4.2 beschrieben – im Schema „Rules“.

Im Folgenden wird noch einmal aufbauend auf Abschnitt 6.4.2 auf Datenqualitätsdimensionen und deren Aggregation eingegangen.

6.5.2 Datenqualitätsdimensionen und Regeln

Die Qualität einer Regel wird im Repository durch ihren Indikator ausgedrückt. Um nun mehrere Regeln zu einem aggregierten Indikator zusammenzufassen, existiert zum einen die Funktion *AggregateIndicator* aus dem Schema „Rules“ und eine weitere Funktion mit gleichem Namen im Schema „Control“. Beide Funktionen sind weitgehend identisch. Allerdings beinhaltet die Funktion im Schema „Rules“ zwei zusätzliche Parameter zum Einschränken eines Zeitraums und liefert entsprechend eine Tabelle mit Datum und Indikator zurück. Der erste Parameter bei beiden Funktionen ist eine Suchbedingung in SQL, die die Regeln einschränkt, die in den aggregierten Indikator einfliessen.

Zur Aggregation von Regeln ist es sinnvoll, über die Kategorie einer Regel die Klassifikation der Regelmenge vorzunehmen. Soll z.B. nur auf „Nützlichkeit“ der Daten überprüft werden, ist es ausreichend für alle Regeln die Merkmale „Vollständigkeit“, „Widerspruchsfreiheit“, „Korrektheit“ u.ä. als eine Kategorie zusammenzufassen. Sollen dagegen auch Aggregationen über die Merkmale erfolgen, so kann die Hierarchie dieser Klassifikation über ein Trennsymbol abgebildet werden. Das folgende erste Beispiel gibt einen aggregierten Indikator für „Nützlichkeit“ zurück und das zweite nur für die Merkmale „Vollständigkeit“ und „Korrektheit“, wenn als Trennzeichen der Punkt verwendet wird.

```
SELECT q.Control.AggregateIndicator( 'Category LIKE ''Nützlichkeit%''' )

SELECT q.Control.AggregateIndicator(
         'Category = ''Nützlichkeit.Vollständigkeit'' OR
          Category = ''Nützlichkeit.Korrektheit''' )
```

Die Aggregation der Regeln zu einem Gesamtindikator erfolgt über das „Simple Additive Weighting“ Verfahren, indem zunächst der Indikator einer Regel mit dem festgelegten Gewicht multipliziert und dann alle Ergebnisse summiert werden.

7. Schlussbemerkung und Kritik

Im Rahmen der Arbeit war zu klären, inwieweit Verfahren zum Verstehen, Verbessern und Steuern der Datenqualität in heutige relationale Datenbankmanagementsysteme (RDBMS) integriert werden können. Dazu wurden zunächst grundlegende Verfahren, die zur Verbesserung der Datenqualität geeignet erscheinen, erläutert und die aktuelle Softwarearchitektur für ein RDBMS beschrieben. Darauf aufbauend wurden drei Architekturansätze zur Einbindung von Datenqualitätsverfahren diskutiert und der Ansatz zur Erweiterung der bestehenden RDBMS-Basisarchitektur mit vorhandenen SQL-Sprachkonstrukten ausgewählt. Nach detaillierten Ausführungen zu dieser Architektur und ihrer Implementierung in ein bestehendes RDBMS als Prototyp, wurden die Konzepte und die Umsetzung hierzu beschrieben.

In der Arbeit wird gezeigt, dass über standardisierte SQL-Sprachkonstrukte viele Verfahren zur Verbesserung der Datenqualität integriert werden können. Gerade die Kombination der SQL mit benutzerdefinierten Funktionen bietet dabei, wie im Prototypen zu sehen, eine leistungsfähige Möglichkeit zur Verbesserung und Analyse von Daten. Positiv hervorzuheben ist auch die Mächtigkeit, das Datenqualitäts-Framework mit Plug-Ins zu erweitern, indem über Reflexion neue und abgeleitete Klassen beliebig integriert werden können.

Generell ist also festzuhalten, dass die Umsetzung der Erweiterung eines RDBMS ausschließlich durch SQL-Sprachkonstrukte gegeben ist. Im Detail ist diese Aussage allerdings unter folgenden Aspekten kritisch zu betrachten:

- Geschäftsregeln
- SQL-Sprachkonstrukte
- Performanz

Geschäftsregeln bilden die Grundlage eines konsistenten korrekten Datenbestandes. Zwar bietet der SQL-Standard über Assertions die Möglichkeit, tabellenübergreifende Bedingungen zu deklarieren und über Trigger auf Veränderungen der Daten direkt zu reagieren, aber eine Abbildung von ECA-Regeln (Event Condition Action)[481], wie sie aktive Datenbanken

[481] Vgl. hierzu: Dittrich, K.R.; Gatziu, S.: Aktive Datenbanksysteme - Konzepte und Mechanismen, 2. Auflage; dpunkt Verlag; Heidelberg

fordern, existiert nicht. Traditionelle RDBMS unterstützen Ereignisse letztendlich nur durch Trigger. Aktive Datenbanken stellen einen ersten Ansatz dar, Regeln in Datenbanken in einer allgemeinen Form zu implementieren. Ein aktives Datenbanksystem ist somit in der Lage „definierbare Situationen in der Datenbank zu erkennen und als Folge davon, bestimmte (ebenfalls definierbare) Reaktionen auszulösen“.[482] Zwar reagieren Trigger auch auf Ereignisse, dies aber nur sehr beschränkt. So ist es z.B. nicht möglich auf zeitliche Ereignisse zu reagieren.[483] Gerade aktive Datenbanksysteme bieten, wie von K.R. Dittrich[484] beschrieben, eine verständliche und einfache Form zur Definition von Regeln.

Die Konzeption und Implementierung des Prototypen, die sich wie in dieser Arbeit am aktuellen SQL-Standard orientiert, verwendet zum Verwalten der Regeln SQL-Routinen und zur Implementierung Trigger und eine zusätzliche Softwarekomponente zur Umsetzung von zeitlichen Ereignissen. Eine intuitive Definition von Regeln ist dadurch nicht unmittelbar möglich. Hier wäre eine Einbettung der Regelverwaltung mit eigenen SQL-Sprachkonstrukten für Regeln sinnvoll.

Neben der Integration der Konzepte aktiver Datenbanksysteme sind zudem die Konzepte von „Business Rules Engines“ bei einer Integration zu berücksichtigen. Anwendungen werden dadurch für das Unternehmen transparenter und Ansatzpunkte zur Verbesserung der Datenqualität sind leichter zu erkennen.[485]

Ein weiterer wesentlicher Kritikpunkt liegt in der Historie von SQL bedingt. Zwar unterstützt der SQL-Standard objektorientierte Paradigmen in SQL selbst, allerdings ist die Einbindung objektorientierter Klassenbibliotheken mit einem großen Umfang an Klassen, Methoden und Eigenschaften, wie sie ein Datenqualitäts-Framework fordert, nicht gegeben. So ist es nicht

---

Schlesinger, M.: ALFRED - Konzepte und Prototyp einer aktiven Schicht zur Automatisierung von Geschäftsregeln, Dissertation; Rechts- und Wirtschaftswissenschaftliche Fakultät der Universität Bern; Bern; 2000
Fritschi, H.: A Component Framework to Construct Active Database Management Systems, Dissertation; Wirtschaftswissenschaftliche Fakultät der Universität Zürich; Zürich; 2002

[482] Dittrich, K.R.; Gatziu, S.: Aktive Datenbanksysteme - Konzepte und Mechanismen, 2. Auflage; dpunkt Verlag; Heidelberg; 2000; S. 7
[483] Schlesinger, M.: ALFRED - Konzepte und Prototyp einer aktiven Schicht zur Automatisierung von Geschäftsregeln, Dissertation; Rechts- und Wirtschaftswissenschaftliche Fakultät der Universität Bern; Bern; 2000; S. 102
[484] Dittrich, K.R.; Gatziu, S.: Aktive Datenbanksysteme - Konzepte und Mechanismen, 2. Auflage; dpunkt Verlag; Heidelberg; 2000
[485] Vgl. hierzu: Chisholm, M.: How to build a Business Rules Engine; Morgan Kaufmann; San Francisco; 2004; S. 443

möglich, einen hierarchischen Namensraum für die Methoden zu bilden.[486] Daneben ist für eine Umsetzung das Überladen und asynchrone Aufrufen von SQL-Routinen sinnvoll. Um ein Datenqualitäts-Framework weitestgehend flexibel aber auch einfach zu gestalten, wäre die Unterstützung variabler Parameterlisten oder benannter Parameter notwendig. Im Prototypen wurde dieses Problem durch die Übergabe einer Zeichenkette gelöst, die eine Auflistung der Parameter mit den Eigenschaftsnamen enthält.

Der dritte kritische Aspekt betrifft einen der wesentlichen Einflussfaktoren, die die Architektur heutiger RDBMS-Implementierungen geprägt hat, die Performanz des Datenbanksystems. Ein Teil der implementierten Datenqualitätsverfahren, vor allem die Verfahren zur Duplikaterkennung, sind sehr rechen- und speicherintensiv. Ein RDBMS sollte hier zum einen die Möglichkeit bieten, Routinen asynchron mit der Möglichkeit von Benachrichtigungen vorzusehen. Zum anderen ist die Forschung gefragt, die einzelnen vorgestellten Verfahren im Detail unter Berücksichtigung der Performanz in ein RDBMS integriert zu entwickeln. Dazu bietet der implementierte Prototyp eine erste Plattform für weitere Untersuchungen. Weitere Forschungen sind in der Berücksichtigung qualitativer Datenqualitätsmerkmale und in der Möglichkeit der Integration dieser in ein RDBMS notwendig. Beispielsweise wäre es denkbar, dass Anwender die Qualität von Daten selbst auf einer Skala bewerten und diese Beurteilung als eine Regel in einen Gesamtindikator mit eingeht.

Letztendlich bleibt festzuhalten, dass eine Erweiterung zwar möglich, aber eine Kombination eines Architekturansatzes aus Erweiterung und Einbettung sinnvoll ist.

Die Arbeit betrachtet nur einen Teil des Themas Datenqualität in Bezug auf Datenbankmanagementsysteme. Datenqualität wird aber auch durch Datenumgebungen bestimmt, die nicht nur das Datenbanksystem selbst, sondern vor allem auch Prozesse, Regeln, Methoden, Richtlinien und Unternehmenskultur beinhalten.[487] Dabei können die Daten nicht in ein fixes Modell oder eine fixe Technik gekapselt werden, da diese letztendlich alle möglichen Facetten der realen Welt abbilden.[488] Das hier vorgestellte Konzept einer Integration bietet

[486] Im Prototypen wurde eine einfache Hierarchie durch die Verwendung von Katalog/Schema sowie Präfixen umgesetzt
[487] Lee, Y.W.; Pipino, L.L.; Funk, J.D.; Wang, R.Y.: Journey to Data Quality; MIT Press; Cambridge, Massachusetts; 2006; S. 79
[488] Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006; S. 235

daher verschiedene Ansätze, dass Framework flexibel zu halten und auch jede Form von Erweiterungen durch Plug-ins zuzulassen.

Probleme der Datenqualität lassen sich jedoch nicht durch Informationstechnik alleine beheben, vielmehr sind Managementansätze notwendig, die die Qualität auf Datenebene als oberstes Ziel betrachten und damit Qualitätsziele, Kennzahlen und deren Einhaltung festlegen. Dazu ist ein Datenqualitätsmanagementsystem einzurichten, das sich in der Organisationsstruktur wiederfindet und vom Management initiiert und „vorgelebt“ wird. Voraussetzung ist, dass die notwendigen Informationen, Methoden, Kennzahlen, Techniken und Werkzeuge vorhanden sind und von den Mitarbeitern gezielt eingesetzt werden können.[489] Gerade die Transparenz der Datenqualität für die Mitarbeiter im Sinne eines Wissensmanagement und auch die Bewertung der Datenqualität durch Mitarbeiter kann entscheidend dazu beitragen, dass Datenqualität miterlebt werden kann. Y.W. Lee und D.M. Strong bringen dieses Wissen der Mitarbeiter auf den Punkt, wenn sie festhalten: „At minimum, data collectors must know what, how and why to collect the data; data custodians must know what, how, and why to store the data; and data consumers must know what, how, and why to use the data”.[490]

Entscheidend für eine Verbesserung der Datenqualität ist nicht ein einziges Vorgehen, sondern vielmehr die Entscheidung für bestimmte Techniken, Methoden, Richtlinien, Aktivitäten usw. abhängig von den Umständen und Zielen, die man mit einer Verbesserung der Daten verfolgt. Nur über eine vollständige Unternehmensrichtlinie kann Datenqualität zum Erfolg führen, die alle Funktionen und Aktivitäten zur Wartung der Daten bestimmt.[491]

Da relationale Datenbankmanagementsysteme einen großen Teil der strukturierten Daten eines Unternehmens speichern, müssen diese flexibler in der Erweiterung werden und Datenqualitätsaspekte verstärkt einbinden. Einer der Pioniere relationaler Datenbanksysteme, J. Gray, und M. Compton konstatieren: “Databases are evolving from SQL engines to data integrators and mediators that offer transactional and nonprocedural access to data in many different forms. This means database systems are effectively becoming database operating

[489] Helfert, M.: Proaktives Datenqualiätsmanagement in Data-Warehouse-Systemen, Dissertation; logos-Verlag; Berlin; 2002; S. 102
[490] Lee, Y.W.; Strong, D.M.: Knowing-Why About Data Processes and Data Quality; erschienen in: Journal of Management Information Systems; Vol. 20, No. 3; 2004; M.E. Sharpe; Armonk, New York, USA; S. 33
[491] Lee, Y.W.; Pipino, L.L.; Funk, J.D.; Wang, R.Y.: Journey to Data Quality; MIT Press; Cambridge, Massachusetts; 2006; S. 172

systems, into which various subsystems and applications can be readily plugged".[492] Nach J. Gray werden sich Datenbanksysteme zukünftig immer mehr zu selbstverwaltenden, selbst-organisierenden und selbstreparierenden Systemen entwickeln.[493] Für ein Datenbanksystem ist dieses aus technischer Sicht heutzutage sicherlich möglich, aber auch für die Daten selbst müssen entsprechende Verfahren entwickelt werden, denn langfristig ist die Qualität der Daten für ein Unternehmen marktentscheidend und nicht die Qualität einer Technik. Diese kann hierbei immer nur unterstützend wirken.

[492] Gray, J.; Compton, M.: A Call To Arms; erschienen in: ACM Queue; Vol. 3, No. 3; ACM; New York, USA; 2005; S. 38
[493] Gray, J.: The Revolution in Database Architecture, Keynote talk; erschienen in: ACM SIGMOD 2004; Paris; 2004; S. 4

A Klassenbibliothek

A.1 Allgemein

Anhang A enthält eine Auflistung aller Klassen, die für die Verwendung vom Framework aus duch SQL-Anweisungen relevant sind. Jede Klasse wird mit einer kurzen Beschreibung und mit dessen öffentlichen Properties, dem Assemblynamen, dem Namensraum, sowie einem kurzen Beispiel dokumentiert. Auf weitere Elemente der Klasse, wie Methoden usw. wird nicht eingegangen, da sie für die Verwendung über SQL nicht von Bedeutung sind. Das Beispiel zu jeder Klasse orientiert sich an einem einfachen Datenmodell, das dem Anhang C entnommen werden kann.

A.2 Basis

A.2.1 Encoder

Namensraum	FreD.Number.Encoder
Assembly	Core
Klasse	DecimalPlacesEncoder

Beschreibung Ermittelt die Anzahl Nachkommastellen einer Fließkommazahl.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden.
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).

Beispiel:

```
SELECT DISTINCT q.Text.Encode(
       ListPrice,
       'FreD.Number.Encoder.DecimalPlacesEncoder',
       'Core', null )
FROM  Sales.Articles
```

Namensraum	FreD.Number.Encoder
Assembly	Core
Klasse	CheckDigitsEncoder

Beschreibung Berechnen einer Prüfziffer auf Modulo-Basis

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden.
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).
Modulo	Konstante, durch die dividiert wird (Default: 10).
Weights	Gewichte pro Ziffernstelle.
Mappings	Umsetzen von Prüfziffern, z.B. errechnet Zahl 10 in X.
SumOfDigits	Bestimmt, ob bei der Berechnung der einzelnen Ziffernstellen aus dem Ergebnis die Quersumme gebildet wird.

Beispiel:

```
-- ISBN-10 Prüfziffer
UPDATE Sales.Articles
SET    ProductNumber = q.Number.Encode( ProductNumber,
       'FreD.Number.Encoder.CheckDigitsEncoder',
       'Core',
       'Weights=10|9|8|7|6|5|4|3|2, Modulo=11, Mappings=10:X'  )
```

Namensraum	FreD.Number.Encoder
Assembly	Core
Klasse	VerhoeffCheckDigitsEncoder

Beschreibung	Berechnen einer Prüfziffer nach dem Verhoeff-Algorithmus (Implementierung nach: J. Kirtland: Identification Numbers and Check Digit Schemes; The Mathematical Association of America; 2001; S. 152ff).

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden.
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).

Beispiel:

```
SELECT ProductNumber,
       q.Number.Encode( ProductNumber,
       'FreD.Number.Encoder.VerhoeffCheckDigitsEncoder',
       'Core', null)
FROM Sales.Articles

SELECT q.Number.Decode(
       '0016342765',
       'FreD.Number.Encoder.VerhoeffCheckDigitsEncoder',
       'Core', null)

SELECT q.Number.Decode( -- Ausnahme wegen falscher Prüfziffer
       '0016342761',
       'FreD.Number.Encoder.VerhoeffCheckDigitsEncoder',
       'Core', null)
```

Namensraum	FreD.Text.Encoder
Assembly	Core
Klasse	GenericTextPreparer

Beschreibung Filtert aus einer Zeichenkette einzelne Zeichen heraus und wandelt die Zeichenkette in Großbuchstaben (abhängig von der Eigenschaft IgnoreCase).

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden.
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Standard: false).
IgnoreCase	Wandelt Zeichenkette in Großbuchstaben um (Default: false).
IgnoreBlank	Filtert Leerzeichen aus (Default: false).
IgnoreNotPrintable	Filtert nicht-druckbare Zeichen aus (Default: true).
IgnoreChars	Ignoriert die in der Eigenschaft übergebenen Zeichen (Default: null).

Beispiel:

```
SELECT Description,
       q.Number.Encode( Description,
       'FreD.Text.Encoder.GenericTextPreparer',
       'Core',
       'IgnoreCase=true, IgnoreBlank=true, IgnoreChars=''AEIOU.,;''' )
FROM Sales.Articles
WHERE Description IS NOT NULL
```

Namensraum	FreD.Text.Encoder
Assembly	Core
Klasse	RemoveStopwordsTextPreparer

Beschreibung Entfernt Stoppwörter aus einer Zeichenkette.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).
Repository	Datenbanktabelle mit Stoppwörtern (Default: „Stopwords“, entspricht Tabelle „q.Repository.stdStopwords“ mit englischsprachigen Stoppwörtern, für deutschsprachige Stoppwörter Eigenschaft auf „StopwordsGerman“ setzen).

Beispiel:

```
SELECT Description,
       q.Number.Encode( Description,
       'FreD.Text.Encoder.RemoveStopwordsTextPreparer',
       'Core', null )
FROM Sales.Articles
WHERE Description IS NOT NULL
```

Namensraum	FreD.Text.Encoder
Assembly	CaumannStemmerTextPreparer
Klasse	CaumannStemmerTextPreparer

Beschreibung	Führt den Stemming-Algorithmus von J. Caumann für deutschsprachigen Text durch.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Standard: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).
Repeat	Legt fest, wie oft der Stemming-Algorithmus für jedes einzelne Wort durchlaufen werden soll (Default: 1).

Beispiel:

```
SELECT Description,
       q.Text.Encode( Description,
       'FreD.Text.Encoder.CaumannStemmerTextPreparer',
       'CaumannStemmerTextPreparer',
       'Repeat=2, Sort=true' )
FROM Sales.Articles
WHERE Description IS NOT NULL
```

Namensraum	FreD.Text.Encoder
Assembly	KrovetzStemmerTextPreparer
Klasse	KrovetzStemmerTextPreparer

Beschreibung Führt den Stemming-Algorithmus von R. Krovetz durch.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).

Beispiel:

```
SELECT Description,
       q.Text.Encode( Description,
       'FreD.Text.Encoder.KrovetzStemmerTextPreparer',
       'KrovetzStemmerTextPreparer',
       null )
FROM Sales.Articles
WHERE Description IS NOT NULL
```

Namensraum	FreD.Text.Encoder
Assembly	LovinsStemmerTextPreparer
Klasse	LovinsStemmerTextPreparer

Beschreibung Führt den Stemming-Algorithmus von J. Lovins durch.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).
Repeat	Legt fest, wie oft der Stemming-Algorithmus für jedes einzelne Wort durchlaufen werden soll (Default: 1).

Beispiel:

```
SELECT Description,
       q.Text.Encode( Description,
       'FreD.Text.Encoder.LovinsStemmerTextPreparer',
       'LovinsStemmerTextPreparer',
       null )
FROM Sales.Articles
WHERE Description IS NOT NULL
```

Namensraum	FreD.Text.Encoder
Assembly	PaiceHuskStemmerTextPreparer
Klasse	PaiceHuskStemmerTextPreparer

Beschreibung Führt den Stemming-Algorithmus von C. Paice und G. Husk durch

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).
Repeat	Legt fest, wie oft der Stemming-Algorithmus für jedes einzelne Wort durchlaufen werden soll (Default: 1).

Beispiel:

```
SELECT Description,
       q.Text.Encode( Description,
       'FreD.Text.Encoder.PaiceHuskStemmerTextPreparer',
       'PaiceHuskStemmerTextPreparer',
       null )
FROM Sales.Articles
WHERE Description IS NOT NULL
```

Namensraum	FreD.Text.Encoder
Assembly	PorterStemmerTextPreparer
Klasse	PorterStemmerTextPreparer

Beschreibung Führt den Stemming-Algorithmus von M.F. Porter durch.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).
Repeat	Legt fest, wie oft der Stemming-Algorithmus für jedes einzelne Wort durchlaufen werden soll (Default: 1).
Country	Legt die Sprache des Textes fest (Default: german, andere Werte: danish, dutch, english, finnish, french, german, italian, norwegian, portuguese, russian, spanish, swedish).

Beispiel:

```
SELECT Description,
       q.Text.Encode( Description,
       'FreD.Text.Encoder.PorterStemmerTextPreparer',
       'PorterStemmerTextPreparer',
       'Country=english' )
FROM sample.Sales.Articles
WHERE Description IS NOT NULL
```

Namensraum	FreD.Text.Encoder
Assembly	UEALiteStemmerTextPreparer
Klasse	UEALiteStemmerTextPreparer

Beschreibung Implementiert den UEA (University of East-Anglia)-Lite Stemmer.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).
Repeat	Legt fest, wie oft der Stemming-Algorithmus für jedes einzelne Wort durchlaufen werden soll (Default: 1).

Beispiel:

```
SELECT Description,
       q.Text.Encode( Description,
       'FreD.Text.Encoder.UEALiteStemmerTextPreparer',
       'UEALiteStemmerTextPreparer',
       null )
FROM sample.Sales.Articles
WHERE Description IS NOT NULL
```

Namensraum	FreD.Text.Encoder
Assembly	Core
Klasse	CharSortPatternEncoder

Beschreibung Gibt die Zeichen einer Zeichenkette sortiert zurück.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).
IgnoreMultipleChar	Berücksichtigt das gleiche mehrfach auftretende Zeichen nur einmal (Default: true).
UpperCase	Konvertiert in Großbuchstaben (Default: true).

Beispiel:

```
SELECT DISTINCT q.Text.Encode(
       Name,
       'FreD.Text.Encoder.CharSortPatternEncoder',
       'Core',
       'IgnoreMultipleChar=true, UpperCase=true' )
FROM  Persons.Customers
```

Namensraum	FreD.Text.Encoder
Assembly	Core
Klasse	DataTypeRecognisedPatternEncoder

Beschreibung Ermittelt den Datentyp eines Wertes.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).
Translate	Gibt einen .NET-Datentyp (Translate=false) oder einen SQL-Datentyp (Translate=true, Default) zurück.

Beispiel:

```
SELECT DISTINCT q.Text.Encode(
       PostalCode,
       'FreD.Text.Encoder.DataTypeRecognisedPatternEncoder',
       'Core',
       'Translate=true' )
FROM  Persons.Customers
```

Namensraum	FreD.Text.Encoder
Assembly	Core
Klasse	NamedEntityPatternEncoder

Beschreibung Erkennt vordefinierte Entitäten aus einer Nachschlageliste, wobei Nachschlagetabellen sowohl normale Nachschlagetexte, aber auch reguläre Ausdrücke und auch kodierte Nachschlagewerte enthalten können.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer)
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).
Repository	Nachschlagetabelle zur Identifikation von Entitäten (Default: NamedEntity, entspricht Tabelle „Repository.stdNamedEntity“).
Encoder	Klassenname eines Encoders, der auf die Zeichenkette angewendet wird, um danach diesen Wert in der Nachschlagetabelle zu suchen.
UnkownEntity	Zeichenkette für einen nicht gefundenen Eintrag (Default: unknown).
ExtractValue	Extrahiert aus einer Zeichenkette nur das semantische Objekt, das in *ExtractValue* angegeben wurde.

Beispiel:

```
SELECT DISTINCT Name,
       q.Text.Encode(Name,
       'FreD.Text.Encoder.NamedEntityPatternEncoder',
       'Core',
       'Encoder=FreD.Text.Encoder.CharSortPatternEncoder' )
FROM  Persons.Customers
WHERE  q.Text.Encode(Name,
       'FreD.Text.Encoder.NamedEntityPatternEncoder',
       'Core',
       'Encoder=FreD.Text.Encoder.CharSortPatternEncoder' ) <>
       '{FirstName} {LastName} '

SELECT DISTINCT Name,
       q.Text.Encode(Name,
       'FreD.Text.Encoder.NamedEntityPatternEncoder',
       'Core',
       'ExtractValue=FirstName' )
FROM  Persons.Customers
```

Namensraum	FreD.Text.Encoder
Assembly	Core
Klasse	OmissionKeyPatternEncoder

Beschreibung Liefert einen Code nach dem Omission Key-Algorithmus von J. Pollock und A. Zamora zurück.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).

Beispiel:

```
SELECT DISTINCT LastName,
       q.Text.Encode( Name,
       'FreD.Text.Encoder.OmissionKeyPatternEncoder',
       'Core',
       null )
FROM  Persons.Customers
ORDER BY 2
```

Namensraum	FreD.Text.Encoder
Assembly	Core
Klasse	SkeletonKeyPatternEncoder

Beschreibung Liefert einen Code nach dem Skeleton Key-Algorithmus von J. Pollock und A. Zamora zurück.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).

Beispiel:

```
SELECT DISTINCT Name,
       q.Text.Encode(Name,
       'FreD.Text.Encoder.SkeletonKeyPatternEncoder',
       'Core',
       null )
FROM  Persons.Customers
ORDER BY 2
```

Namensraum	FreD.Text.Encoder
Assembly	Core
Klasse	ReverseOrderPatternEncoder

Beschreibung Die Zeichen einer Zeichenkette werden in umgekehrter Reihenfolge zurückgegeben.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).

Beispiel:

```
SELECT DISTINCT Name,
       q.Text.Encode(Name,
       'FreD.Text.Encoder.ReverseOrderPatternEncoder',
       'Core',
       null )
FROM  Persons.Customers
```

Namensraum	FreD.Text.Encoder
Assembly	Core
Klasse	SimpleOrderPatternEncoder

Beschreibung Ersetzt jeden Groß-, Kleinbuchstaben, jede Zahlen und andere Zeichen durch ein gemeinsames Kürzel.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).
LowerLetter	Zeichenkette, die Kleinbuchstaben ersetzt (Default: ‚A').
UpperLetter	Zeichenkette, die Großbuchstaben ersetzt (Default: ‚a').
Digit	Zeichenkette, die Zahlen ersetzt (Default: ‚9').
Other	Zeichenkette, die alle anderen Zeichen außer Groß-/Kleinbuchstaben und Zahlen ersetzt (Default: ‚*').

Beispiel:

```
SELECT q.Text.Encode( PostalCode,
       'FreD.Text.Encoder.SimplePatternEncoder',
       'Core',
       'Other='),
       COUNT(*)
FROM  Persons.Customers
GROUP BY q.Text.Encode( PostalCode,
       'FreD.Text.Encoder.SimplePatternEncoder',
       'Core',
       'Other=' )
```

Namensraum	FreD.Text.Encoder
Assembly	Core
Klasse	Adler32HashingEncoder

Beschreibung Berechnet den Hashwert nach dem Adler32-Algorithmus.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).

Beispiel:

```
SELECT DISTINCT q.Text.Encode( PostalCode,
       'FreD.Text.Encoder.Adler32HashingEncoder',
       'Core',
       NULL)
FROM  Persons.Customers
```

Namensraum	FreD.Text.Encoder
Assembly	Core
Klasse	SumHashingEncoder

Beschreibung Berechnet einen Hashwert, indem es die Quersumme bildet.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).

Beispiel:

```
SELECT DISTINCT q.Text.Encode( PostalCode,
       'FreD.Text.Encoder.SumHashingEncoder',
       'Core',
       NULL)
FROM  Persons.Customers
```

Namensraum	FreD.Text.Encoder
Assembly	Core
Klasse	CaverPhoneticEncoder

Beschreibung Erstellt den phonetischen Code nach dem Caverphone-Algorithmus

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).
MaxLength	Maximaler Länge des phonetischen Codes (Default: 4).

Beispiel:

```
SELECT DISTINCT q.Text.Encode(Name,
       'FreD.Text.Encoder.CaverPhoneticEncoder',
       'Core',
       'MaxLength=6')
FROM  Persons.Customers
```

Namensraum	FreD.Text.Encoder
Assembly	Core
Klasse	ColognePhoneticEncoder

Beschreibung Erstellt den Kölner Phonetik-Code nach H.-J. Postel.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).
MaxLength	Maximaler Länge des phonetischen Codes (Default: 4).

Beispiel:

```
SELECT DISTINCT Name, q.Text.Encode( Name,
       'FreD.Text.Encoder.ColognePhoneticEncoder',
       'Core',
       null)
FROM  Persons.Customers
ORDER BY 2
```

Namensraum	FreD.Text.Encoder
Assembly	Core
Klasse	DaitchMokotoffPhoneticEncoder

Beschreibung Erweiterung des Soundex-Codes von R. Daitch und G. Mokotoff.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).
MaxLength	Maximaler Länge des phonetischen Codes (Default: 6).
UseAlternateCode	Falls vorhanden, wird ein zweiter alternativer Code zurückgegeben (Default: false).

Beispiel:

```
SELECT DISTINCT Name, q.Text.Encode( Name,
       'FreD.Text.Encoder.DaitchMokotoffPhoneticEncoder',
       'Core',
       'UseAlternateCode=false')
FROM  Persons.Customers
ORDER BY 2
```

Namensraum	FreD.Text.Encoder
Assembly	Core
Klasse	IBMAlphaSearchPhoneticEncoder

Beschreibung Erzeugt einen phonetischen Code nach dem „IBM Alpha Inquiry System Personal Name Encoding“-Algorithmus. Ausnahmen für einen zweiten und dritten Durchgang wurden nicht implementiert.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).
MaxLength	Maximaler Länge des phonetischen Codes (Default: 14).

Beispiel:

```
SELECT DISTINCT Name, q.Text.Encode( Name,
       'FreD.Text.Encoder.IBMAlphaSearchPhoneticEncoder',
       'Core',
       null)
FROM  Persons.Customers
ORDER BY 2
```

Namensraum	FreD.Text.Encoder
Assembly	Core
Klasse	NysiisPhoneticEncoder

Beschreibung Erzeugt einen phonetischen Code nach dem NYSIIS-Algorithmus („New York State Identification and Intelligence System“). Alternative erweiterte Version, siehe http://www.dropby.com/NYSIIS.html.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).
MaxLength	Maximaler Länge des phonetischen Codes (Default: 4).

Beispiel:

```
SELECT DISTINCT Name, q.Text.Encode( Name,
       'FreD.Text.Encoder.NysiisPhoneticEncoder',
       'Core',
       null )
FROM  Persons.Customers
ORDER BY 2
```

Namensraum	FreD.Text.Encoder
Assembly	Core
Klasse	ONCAPhoneticEncoder

Beschreibung	Erzeugt einen phonetischen Code nach dem ONCA („Oxford Name Compression Algorithm“), der NYSIIS und Soundex kombiniert.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).
MaxLength	Maximaler Länge des phonetischen Codes (Default: 4).

Beispiel:

```
SELECT DISTINCT Name, q.Text.Encode( Name,
       'FreD.Text.Encoder.ONCAPhoneticEncoder',
       'Core',
       null )
FROM  Persons.Customers
ORDER BY 2
```

Namensraum	FreD.Text.Encoder
Assembly	Core
Klasse	PhonemPhoneticEncoder

Beschreibung Erzeugt einen phonetischen Code nach dem PHONEM-Algorithmus von G. Wilde und C. Meyer.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).
MaxLength	Maximaler Länge des phonetischen Codes (Default: 4).

Beispiel:

```
SELECT DISTINCT Name, q.Text.Encode( Name,
       'FreD.Text.Encoder.PhonemPhoneticEncoder',
       'Core',
       null )
FROM  Persons.Customers
ORDER BY 2
```

Namensraum	FreD.Text.Encoder
Assembly	Core
Klasse	PhonexPhoneticEncoder

Beschreibung	Erzeugt einen phonetischen Code nach dem Phonex-Algorithmus von A.J. Lait und B. Randell[494], einer Variation des Soundex-Algorithmus.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).
MaxLength	Maximaler Länge des phonetischen Codes (Default: 4).

Beispiel:

```
SELECT DISTINCT Name, q.Text.Encode( Name,
       'FreD.Text.Encoder.PhonemPhoneticEncoder',
       'Core',
       null )
FROM  Persons.Customers
ORDER BY 2
```

[494] Siehe Lait, A.J., Randell, B.: An Assessment of Name Matching Algorithms, Technical Report; Department of Computing Science, University of Newcastle upon Tyne; Newcastle; 1993

Namensraum	FreD.Text.Encoder
Assembly	Core
Klasse	RethSchekPhoneticEncoder

Beschreibung Erzeugt einen phonetischen Code für deutsche Namen nach einem Algorithmus von H.-P. von Reth und H.-J. Schek.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).
MaxLength	Maximaler Länge des phonetischen Codes (Default: 4).

Beispiel:

```
SELECT DISTINCT Name, q.Text.Encode( Name,
       'FreD.Text.Encoder.RethSchekPhoneticEncoder',
       'Core',
       null )
FROM  Persons.Customers
ORDER BY 2
```

Namensraum	FreD.Text.Encoder
Assembly	Core
Klasse	SoundexPhoneticEncoder

Beschreibung Erzeugt einen phonetischen Code nach dem Soundex-Algorithmus von M. Odell und R. Russel.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).
MaxLength	Maximaler Länge des phonetischen Codes (Default: 4).

Beispiel:

```
SELECT DISTINCT Name, q.Text.Encode( Name,
       'FreD.Text.Encoder.RethSchekPhoneticEncoder',
       'Core',
       null )
FROM  Persons.Customers
ORDER BY 2
```

Namensraum	FreD.Text.Encoder
Assembly	Core
Klasse	SoundexPhoneticExtendedEncoder

Beschreibung Erzeugt einen phonetischen Code nach dem Soundex-Algorithmus von M. Odell und R. Russel. Außerdem wird das erste Zeichen auch kodiert und es existieren zwei zusätzliche Eigenschaften.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden (Default: GenericTokenizer).
Sort	Sortiert die Tokens aufsteigend, bevor sie zu einem Wert konkateniert werden (Default: false).
MaxLength	Maximaler Länge des phonetischen Codes (Default: 4).
FillCode	Ergänzt den Code um das Zeichen ‚0' am Ende, wenn weniger als MaxLength (default: true).
Mapping	26-Zeichen lange Zeichenkette mit dem Mapping der Buchstaben von A-Z (default: „0123012-02245501262301-2-2").

Beispiel:

```
SELECT DISTINCT Name, q.Text.Encode( Name,
       'FreD.Text.Encoder.SoundexPhoneticExtendedEncoder',
       'Core',
       'FillCode=false, Mapping=*123*12-*22455*12623*1-2-2' )
FROM  Persons.Customers
ORDER BY 2
```

A.2.2 Matching

Namensraum	FreD.Date.Matching
Assembly	Core
Klasse	DateMatching

Beschreibung Vergleicht zwei Datumswerte und gibt deren Ähnlichkeit als Wert zwischen 0 und 1 zurück.

Eigenschaften

MaxTimeSpan	Zeitintervall, der als Basis zur Berechnung des Ähnlichkeitswertes nach folgender Formel verwendet wird: Abs(Datum1 – Datum2) / MaxTimeSpan (Default: 7 Tage).

Beispiel:

```
SELECT OrderDate, ShipDate,
       Date.Match( OrderDate, ShipDate,
       'FreD.Date.Matching.DateMatching',
       'Core',
       'MaxTimeSpan=14.00:00:00' )
FROM Sales.Orders
```

Namensraum	FreD.Distance.Matching
Assembly	Core
Klasse	GeoDistanceMatching

Beschreibung Ermittelt die Entfernung zwischen zwei Postleitzahlen über deren Geodaten (Längen-/Breitengrad) und berechnet daraufhin die Ähnlichkeit zwischen diesen beiden.

Eigenschaften

MaxDistance	Distanz in Kilometern, die als Basis zur Berechnung des Ähnlichkeitswertes nach folgender Formel verwendet wird: Distanz zwischen den beiden Postleitzahlen / MaxDistance (Default: 100).
Country	Land für die übergebenen Postleitzahlen (Default: DE, andere Werte US bzw. distinkte Werte in Spalte Country der Tabelle Repository.sysGeoData).

Beispiel:

```
  SELECT a.CustomerId, b.CustomerId, a.Name, b.Name, a.PostalCode,
b.PostalCode,
         q.Distance.Match( a.PostalCode, b.PostalCode,
                           'FreD.Distance.Matching.GeoDistanceMatching',
                           'Core',
                           'MaxDistance=50, Country=US' )
  FROM   Persons.Customers a, Persons.Customers b
  WHERE  q.Distance.Match( a.PostalCode, b.PostalCode,
                           'FreD.Distance.Matching.GeoDistanceMatching',
                           'Core',
                           'MaxDistance=50, Country=US' ) > 0.0 AND
                           a.CustomerId > b.CustomerId
```

Namensraum	FreD.Number.Matching
Assembly	Core
Klasse	EuclidianDistanceMatching

Beschreibung Ermittelt die euklidische Distanz zwischen den übergebenen Zahlen und errechnet das entsprechende Ähnlichkeitsmaß zwischen 0 und 1.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token, die einzeln kodiert und dann konkateniert werden.

Beispiel:

```
SELECT a.ArticleId, b.ArticleId, a.Name, b.Name,
       a.ListPrice, b.ListPrice,
       q.Number.Match( a.ListPrice, b.ListPrice,
                       'FreD.Number.Matching.EuclidianDistanceMatching',
                       'Core', null )
FROM   Sales.Articles a, Sales.Articles b
WHERE  q.Number.Match( a.ListPrice, b.ListPrice,
                       'FreD.Number.Matching.EuclidianDistanceMatching',
                       'Core', null ) > 0.8 AND
       a.ArticleId > b.ArticleId
```

Namensraum	FreD.Number.Matching
Assembly	Core
Klasse	SimpleNumberMatching

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zahlen als Wert zwischen 0 und 1.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.
MaxDistance	Maximale Distanz zwischen zwei Zahlen, die als Basis zur Berechnung des Ähnlichkeitswertes nach folgender Formel verwendet wird: Abs(Zahl1 – Zahl2) / MaxDistance (Default: 1).

Beispiel:

```
SELECT a.ArticleId, b.ArticleId, a.Name, b.Name,
       a.ListPrice, b.ListPrice,
       q.Number.Match( a.ListPrice, b.ListPrice,
                       'FreD.Number.Matching.SimpleNumberMatching',
                       'Core',
                       'MaxDistance=100.0' )
FROM   Sales.Articles a, Sales.Articles b
WHERE  q.Number.Match( a.ListPrice, b.ListPrice,
                       'FreD.Number.Matching.SimpleNumberMatching',
                       'Core', 'MaxDistance=100.0' ) > 0.8 AND
       a.ArticleId > b.ArticleId
```

Namensraum	FreD.Text.Matching
Assembly	Core
Klasse	ExactTextMatching

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten zeichenbasiert. Gibt bei exakter Übereinstimmung 1, ansonsten 0 zurück.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.
StartIndex	Startposition, ab der der Vergleich beginnen soll.

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.ExactTextMatching',
                     'Core',
                     'StartIndex=2' )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.ExactTextMatching',
                     'Core',
                     'StartIndex=2' ) = 1.0 AND
       a.CustomerId > b.CustomerId
```

Namensraum	FreD.Text.Matching
Assembly	Core
Klasse	HammingApproximateMatching

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten zeichenbasiert nach dem Algorithmus zur Berechnung der Hamming-Distanz als Wert zwischen 1 und 0.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.HammingApproximateMatching',
                     'Core', null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.HammingApproximateMatching',
                     'Core', null ) > 0.9 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

Namensraum	FreD.Text.Matching
Assembly	Core
Klasse	HirschbergApproximateMatching

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten zeichenbasiert nach dem LCS-Algorithmus von Hirschberg als Wert zwischen 1 und 0.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.HirschbergApproximateMatching',
                     'Core', null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.HirschbergApproximateMatching',
                     'Core', null ) > 0.9 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

Namensraum	FreD.Text.Matching
Assembly	Core
Klasse	JaroApproximateMatching

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten zeichenbasiert nach dem Algorithmus von Jaro als Wert zwischen 1 und 0.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                'FreD.Text.Matching.JaroApproximateMatching',
                'Core',
                '{Tokenizer=FreD.Text.Tokenizer.GenericTokenizer} )' )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                'FreD.Text.Matching.JaroApproximateMatching',
                'Core',
                '{Tokenizer=FreD.Text.Tokenizer.GenericTokenizer}' ) > 0.9
               AND a.CustomerId > b.CustomerId
ORDER BY 5
```

Namensraum	FreD.Text.Matching
Assembly	Core
Klasse	JaroWinklerApproximateMatching

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten zeichenbasiert nach dem angepassten Algorithmus von Jaro als Wert zwischen 1 und 0, indem die Übereinstimmung der ersten n Zeichen berücksichtigt wird.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.
CommonPrefixLength	Anzahl zu überprüfender Zeichen am Anfang der Zeichenkette (Default: 4). Die Anzahl der Übereinstimmungen geht als Korrekturfaktor in das Ähnlichkeitsmaß ein.

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.JaroWinklerApproximateMatching',
                     'Core',
                     'CommonPrefixLength=5' )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.JaroWinklerApproximateMatching',
                     'Core',
                     'CommonPrefixLength=5' ) > 0.9 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

Namensraum	FreD.Text.Matching
Assembly	Core
Klasse	JaroWinklerLynchApproximateMatching

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten zeichenbasiert nach einem angepassten Algorithmus von Jaro/Winkler als Wert zwischen 1 und 0.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.
CommonPrefixLength	Anzahl zu überprüfender Zeichen am Anfang der Zeichenkette (Default: 4). Die Anzahl der Übereinstimmungen geht als Korrekturfaktor in das Ähnlichkeitsmaß ein.

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                'FreD.Text.Matching.JaroWinklerLynchApproximateMatching',
                'Core',
                'CommonPrefixLength=5' )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                'FreD.Text.Matching.JaroWinklerLynchApproximateMatching',
                'Core',
                'CommonPrefixLength=5' ) > 0.9 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

Namensraum	FreD.Text.Matching
Assembly	Core
Klasse	LevenshteinApproximateMatching

Beschreibung Ermittelt die Distanz zwischen zwei Zeichenketten zeichenbasiert nach dem Algorithmus von V. Levenshtein und F. Damerau und errechnet daraus die Ähnlichkeit als Wert zwischen 0 und 1.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.LevenshteinApproximateMatching',
                     'Core',
                     null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.LevenshteinApproximateMatching',
                     'Core',
                     null ) > 0.9 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

Namensraum	FreD.Text.Matching
Assembly	Core
Klasse	LevenshteinHjelmqvistApproximateMatching

Beschreibung Ermittelt die Lebenshtein-Distanz zwischen zwei Zeichenketten zeichenbasiert nach dem performanteren Algorithmus von S. Hjelmquist und errechnet daraus die Ähnlichkeit als Wert zwischen 0 und 1 (siehe http://www.codeproject.com/KB/recipes/Levenshtein.aspx).

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
          'FreD.Text.Matching.LevenshteinHjelmqvistApproximateMatching',
          'Core',
          null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
          'FreD.Text.Matching.LevenshteinHjelmqvistApproximateMatching',
          'Core',
          null ) > 0.9 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

Namensraum	FreD.Text.Matching
Assembly	Core
Klasse	Sift3ApproximateMatching

Beschreibung Ermittelt die Distanz zwischen zwei Zeichenketten zeichenbasiert nach dem Sift3-Algorithmus und errechnet daraus die Ähnlichkeit als Wert zwischen 0 und 1 (siehe http://siderite.blogspot.com/2007/04/super-fast-and-accurate-string-distance.html).

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.Sift3ApproximateMatching',
                     'Core',
                     null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.Sift3ApproximateMatching',
                     'Core',
                     null ) > 0.9 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

Namensraum	FreD.Text.Matching
Assembly	Core
Klasse	TypewriterApproximateMatching

Beschreibung Ermittelt ein zeichenbasiertes Distanzmaß abhängig von der Entfernung unterschiedlicher Zeichen auf der Tastatur und errechnet daraus die Ähnlichkeit als Wert zwischen 0 und 1.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.
ThresholdDistance	Ist dieser Wert < 0, dann wird die Anzahl an Tasten zwischen den beiden unterschiedlichen Zeichen auf der Tastatur verwendet. Andernfalls wird überprüft, ob ThresholdDistance größer als der Tastaturabstand ist und gegebenenfalls die maximale Diszanz (errechnet sich aus dem Tastaturlayout) verwendet (Default: 4).
TypewriterLayout	Zeichenkette die das Tastaturlayout beschreibt. Das ‚|'-Zeichen trennt die einzelnen Tastaturreihen (Default: „1234567890ß|QWERTZUIOPÜ|ASDFGHJKLÖÄ|YXCVBNM;:_“).

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.TypewriterApproximateMatching',
                     'Core',
                     'ThresholdDistance=10' )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.TypewriterApproximateMatching',
                     'Core',
                     'ThresholdDistance=10' ) > 0.9 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

Namensraum	FreD.Text.Matching
Assembly	Core
Klasse	CityBlockApproximateMatching

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten tokenbasiert auf Grundlage gleicher und ungleicher Token als Wert zwischen 0 und 1 über die City-Block-Distanz.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.
TokenizerTokenBased	Tokenizer zum Aufsplitten in mehrere Token zur Berechnung tokenbasierter Vergleichsalgorithmen (Default: GenericTokenizer).

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.CityBlockApproximateMatching',
                     'Core',
                     null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.CityBlockApproximateMatching',
                     'Core',
                     null ) >= 0.5 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

Namensraum	FreD.Text.Matching
Assembly	Core
Klasse	CosineApproximateMatching

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten tokenbasiert auf Grundlage gleicher und ungleicher Token als Wert zwischen 0 und 1 über das Kosinus-Maß.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.
TokenizerTokenBased	Tokenizer zum Aufsplitten in mehrere Token zur Berechnung tokenbasierter Vergleichsalgorithmen (Default: GenericTokenizer).

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                'FreD.Text.Matching.CosineApproximateMatching',
                'Core',
                '{TokenizerTokenBased=FreD.Text.Tokenizer.NGramTokenizer,
                NGramSize=3}' )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                'FreD.Text.Matching.CosineApproximateMatching',
                'Core',
                '{TokenizerTokenBased=FreD.Text.Tokenizer.NGramTokenizer,
                NGramSize=3}' ) >= 0.5 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

Namensraum	FreD.Text.Matching
Assembly	Core
Klasse	DiceApproximateMatching

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten tokenbasiert auf Grundlage gleicher und ungleicher Token als Wert zwischen 0 und 1 über das Dice-Maß.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.
TokenizerTokenBased	Tokenizer zum Aufsplitten in mehrere Token zur Berechnung tokenbasierter Vergleichsalgorithmen (Default: GenericTokenizer).

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.DiceApproximateMatching',
                     'Core',
                     '{Tokenizer=FreD.Text.Tokenizer.NGramTokenizer,
                      NGramSize=1}' )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                     'FreD.Text.Matching.DiceApproximateMatching',
                     'Core',
                     '{Tokenizer=FreD.Text.Tokenizer.NGramTokenizer,
                      NGramSize=1}' ) >= 0.8 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

Namensraum	FreD.Text.Matching
Assembly	Core
Klasse	EuclidianDistanceApproximateMatching

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten tokenbasiert auf Grundlage gleicher und ungleicher Token als Wert zwischen 0 und 1 über über das Euklidische Distanzmaß.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.
TokenizerTokenBased	Tokenizer zum Aufsplitten in mehrere Token zur Berechnung tokenbasierter Vergleichsalgorithmen (Default: GenericTokenizer).

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
           'FreD.Text.Matching.EuclidianDistanceApproximateMatching',
           'Core',
           null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
           'FreD.Text.Matching.EuclidianDistanceApproximateMatching',
           'Core',
           null ) > 0 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

Namensraum	FreD.Text.Matching
Assembly	Core
Klasse	FellegiSunterApproximateMatching

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten tokenbasiert auf Grundlage gleicher und ungleicher Token als Wert zwischen 0 und 1 über den Algorithmus zur Erkennung von Duplikaten nach Fellegi und Sunter.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.
TokenizerTokenBased	Tokenizer zum Aufsplitten in mehrere Token zur Berechnung tokenbasierter Vergleichsalgorithmen (Default: GenericTokenizer).
ScoreMatch	Wert zwischen 0 und 1, der bestimmt, ab wann zwei Token als identisch angesehen werden (Default: 0,8).
Comparator	Zeichenkettenvergleichsalgorithmus zum Vergleich der einzelnen Token (Default: JaroApproximateMatching).
FrequencyTable	Verweis auf Tabelle im Repository mit dem Präfix „frq“ mit den relativen Häufigkeiten einzelner Tokens (kann mit der SQL-Prozedur *Tools.BuildFrequencyTable* erzeugt werden).

Beispiel:

```
EXEC q.Tools.BuildFrequencyTable
     'FrequencySample_Customers',
     'Sample.Persons.Customers',
     'Name', null, 1, 1

SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
   q.Text.Match( a.Name, b.Name,
   'FreD.Text.Matching.FellegiSunterApproximateMatching',
   'Core',
   'FrequencyTable=FrequencySample_Customers,
   {Comparator=FreD.Text.Matching.JaroWinklerLynchApproximateMatching}' )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
   'FreD.Text.Matching.FellegiSunterApproximateMatching',
   'Core',
   'FrequencyTable=FrequencySample_Customers,
   {Comparator=FreD.Text.Matching.JaroWinklerLynchApproximateMatching}')
        >= 0 AND a.CustomerId > b.CustomerId AND a.Name <> b.Name
ORDER BY 5
```

Namensraum	FreD.Text.Matching
Assembly	Core
Klasse	JaccardApproximateMatching

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten tokenbasiert auf Grundlage gleicher und ungleicher Token als Wert zwischen 0 und 1 über über das Jaccard-Maß.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.
TokenizerTokenBased	Tokenizer zum Aufsplitten in mehrere Token zur Berechnung tokenbasierter Vergleichsalgorithmen (Default: GenericTokenizer).

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                 'FreD.Text.Matching.JaccardApproximateMatching',
                 'Core',
                 null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                 'FreD.Text.Matching.JaccardApproximateMatching',
                 'Core',
                 null ) > 0 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

Namensraum	FreD.Text.Matching
Assembly	Core
Klasse	MatchingCoefficientApproximateMatching

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten tokenbasiert auf Grundlage gleicher und ungleicher Token als Wert zwischen 0 und 1 über über das Matching Coefficient Maß.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.
TokenizerTokenBased	Tokenizer zum Aufsplitten in mehrere Token zur Berechnung tokenbasierter Vergleichsalgorithmen (Default: GenericTokenizer).

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
            'FreD.Text.Matching.MatchingCoefficientApproximateMatching',
            'Core',
            null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
            'FreD.Text.Matching.MatchingCoefficientApproximateMatching',
            'Core',
            null ) > 0 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

Namensraum	FreD.Text.Matching
Assembly	Core
Klasse	OverlapApproximateMatching

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten tokenbasiert auf Grundlage gleicher und ungleicher Token als Wert zwischen 0 und 1 über über das Overlap-Maß.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.
TokenizerTokenBased	Tokenizer zum Aufsplitten in mehrere Token zur Berechnung tokenbasierter Vergleichsalgorithmen (Default: GenericTokenizer).

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                 'FreD.Text.Matching.OverlapApproximateMatching',
                 'Core',
                 null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                 'FreD.Text.Matching.OverlapApproximateMatching',
                 'Core',
                 null ) > 0 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

Namensraum	FreD.Text.Matching
Assembly	Core
Klasse	SoftTFIDFApproximateMatching

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten tokenbasiert auf Grundlage gleicher und ungleicher Token als Wert zwischen 0 und 1 über den TF-IDF-Algorithmus.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.
TokenizerTokenBased	Tokenizer zum Aufsplitten in mehrere Token zur Berechnung tokenbasierter Vergleichsalgorithmen (Default: GenericTokenizer)
ScoreMatch	Wert zwischen 0 und 1, der bestimmt, ab wann zwei Token als identisch angesehen werden (Default: 0,8).
Comparator	Zeichenkettenvergleichsalgorithmus zum Vergleich der einzelnen Token (Default: JaroApproximateMatching).
FrequencyTable	Verweis auf Tabelle im Repository mit dem Präfix „frq“ mit den relativen Häufigkeiten einzelner Tokens (kann mit der SQL-Prozedur *Tools.BuildFrequencyTable* erzeugt werden).

Beispiel:

```
EXEC q.Tools.BuildFrequencyTable
   'FrequencySample_Customers',
   'Sample.Persons.Customers',
   'Name', null, 1, 1

SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
         'FreD.Text.Matching.SoftTFIDFApproximateMatching',
         'Core',
         'FrequencyTable=FrequencySample_Customers' )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
         'FreD.Text.Matching.SoftTFIDFApproximateMatching',
         'Core',
         'FrequencyTable=FrequencySample_Customers' ) >= 0.8 AND
         a.CustomerId > b.CustomerId AND a.Name <> b.Name
ORDER BY 5
```

Namensraum	FreD.Text.Matching
Assembly	CompressApproximateMatching
Klasse	CompressApproximateMatching

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten zeichenbasiert als Wert zwischen 0 und 1, indem die Zeichenketten verglichen werden, nachdem diese einen Komprimierungs-algorithmus durchlaufen haben.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                 'FreD.Text.Matching.CompressApproximateMatching',
                 'CompressApproximateMatching',
                 null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                 'FreD.Text.Matching.CompressApproximateMatching',
                 'CompressApproximateMatching',
                 null ) > 0.8 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

Namensraum	FreD.Text.Matching
Assembly	DistanceBasedApproximateMatching
Klasse	NeedlemanWunschApproximateMatching

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten zeichenbasiert als Wert zwischen 0 und 1 nach dem Needleman-Wunsch-Distanzmaß.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                 'FreD.Text.Matching.NeedlemanWunschApproximateMatching',
                 'DistanceBasedApproximateMatching',
                 null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                 'FreD.Text.Matching.NeedlemanWunschApproximateMatching',
                 'DistanceBasedApproximateMatching',
                 null ) > 0.8 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

Namensraum	FreD.Text.Matching
Assembly	DistanceBasedApproximateMatching
Klasse	SmithWatermanApproximateMatching

Beschreibung Ermittelt die Ähnlichkeit zwischen zwei Zeichenketten zeichenbasiert als Wert zwischen 0 und 1 nach dem Smith-Waterman-Distanzmaß.

Eigenschaften

Tokenizer	Tokenizer zum Aufsplitten in mehrere Token. Dabei wird der Ähnlichkeitswert jedes Tokens mit jedem anderen berechnet und der größte Wert jeweils summiert. Die Summe wird durch die höhere Anzahl an Tokens dividiert.

Beispiel:

```
SELECT a.CustomerId, b.CustomerId, a.Name, b.Name,
       q.Text.Match( a.Name, b.Name,
                 'FreD.Text.Matching.SmithWatermanApproximateMatching',
                 'DistanceBasedApproximateMatching',
                 null )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Text.Match( a.Name, b.Name,
                 'FreD.Text.Matching.SmithWatermanApproximateMatching',
                 'DistanceBasedApproximateMatching',
                 null ) > 0.8 AND
       a.CustomerId > b.CustomerId
ORDER BY 5
```

A.2.3 Tokenizer

Namensraum	FreD.Text.Tokenizer
Assembly	Core
Klasse	GenericTokenizer

Beschreibung Trennt eine Zeichenkette in einzelne Tokens auf, indem es eine Liste von Trennzeichen verwendet.

Eigenschaften

Separators	Trennzeichen, die bestimmen, an welchen Stellen die Zeichenkette getrennt wird (Default: „ !\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~§°\t\n“).

Beispiel:

```
SELECT Name,
       q.Text.Tokenize( Name, ' | ',
                        'FreD.Text.Tokenizer.GenericTokenizer',
                        'Core', null )
FROM   Persons.Customers
```

Namensraum	FreD.Text.Tokenizer
Assembly	Core
Klasse	NGramTokenizer

Beschreibung	Trennt eine Zeichenkette in einzelne Tokens auf, indem die Zeichenkette in gleich lange Token der Länge N aufgeteilt werden.

Eigenschaften

NGramSize	Länge der Token (Default: 3).
NormalizeHeadTail	Berücksichtigt Anfang und Ende einer Zeichenkette (Default: true)
PreTokenizer	Tokenizer, der vor Erstellung der N-Gram's auf die Zeichenkette angewendet wird.

Beispiel:

```
SELECT Name,
       q.Text.Tokenize( Name, ' | ',
                 'FreD.Text.Tokenizer.NGramTokenizer',
                 'Core',
                 'NGramSize=2,
                  NormalizeHeadTail=false,
                  {PreTokenizer=FreD.Text.Tokenizer.GenericTokenizer}' )
FROM   Persons.Customers
```

Namensraum	FreD.Text.Tokenizer
Assembly	Core
Klasse	NGramThresholdTokenizer

Beschreibung	Bildet aus N-Gram-Token der Länge N neue zusammengesetzte Token, deren Anzahl von einem Schwellwert abhängt.

Eigenschaften

Threshold	Schwellwert, der die Anzahl zusammengesetzte Token bestimmt (Default: 0,8).
NGramSize	Länge der Token (Default: 3).
NormalizeHeadTail	Berücksichtigt Anfang und Ende einer Zeichenkette (Default: true)
PreTokenizer	Tokenizer, der vor Erstellung der N-Gram's auf die Zeichenkette angewendet wird.

Beispiel:

```
SELECT Name,
       q.Text.Tokenize( Name, ' | ',
                 'FreD.Text.Tokenizer.NGramThresholdTokenizer',
                 'Core',
                 'Threshold=0.9,
                  NGramSize=2,
                  {PreTokenizer=FreD.Text.Tokenizer.GenericTokenizer}' )
FROM   Persons.Customers
```

Namensraum	FreD.Text.Tokenizer
Assembly	Core
Klasse	SuffixArrayTokenizer

Beschreibung Trennt eine Zeichenkette in einzelne Tokens auf, indem die Suffixe einer Zeichenkette bis zu einer minimalen Länge von N gebildet werden.

Eigenschaften

MinSuffixLength	Bestimmt die minimale Länge eines Token (Default: 3)

Beispiel:

```
SELECT Name,
       q.Text.Tokenize( Name, ' | ',
                 'FreD.Text.Tokenizer.SuffixArrayTokenizer',
                 'Core',
                 'MinSuffixLength=10' )
FROM   Persons.Customers
```

A.3 Verstehen

A.3.1 Daten- und Schemaeigenschaften

Namensraum	FreD.Understand.Properties
Assembly	Core
Klasse	CountNotNullProperty
Beschreibung	Ermittelt die Anzahl an NOT NULL Einträgen.
Eigenschaften	
	keine

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
             '{FreD.Understand.Properties.CountNotNullProperty|Core}'+
             COALESCE( Color, '{NULL}') )
FROM Sales.Articles
```

Namensraum	FreD.Understand.Properties
Assembly	Core
Klasse	CountNullProperty

Beschreibung Ermittelt die Anzahl an NULL Einträgen.

Eigenschaften

keine

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
             '{FreD.Understand.Properties.CountNullProperty|Core}'+
             COALESCE( Color, '{NULL}') )
FROM Sales.Articles
```

Namensraum	FreD.Understand.Properties
Assembly	Core
Klasse	DataTypeRecognisedProperty

Beschreibung	Ermittelt den Datentyp einer Tabellenspalte anhand der Dateninhalte.

Eigenschaften

Threshold	Ausgabe als SQL-Datentyp (Default: true), ansonsten als .NET-Datentyp.

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
          '{FreD.Understand.Properties.DataTypeRecognisedProperty|
           Core|Translate=true}'+
          COALESCE( ProductNumber, '{NULL}') )
FROM Sales.Articles
```

Namensraum	FreD.Understand.Properties
Assembly	Core
Klasse	MaxProperty

Beschreibung Ermittelt die n größten Werte einer Spalte.

Eigenschaften

MaxCount	Anzahl der maximalen Werte, die zurückgegeben werden (Default: 5).

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
             '{FreD.Understand.Properties.MaxProperty|Core|MaxCount=10}'+
             COALESCE( ProductNumber, '{NULL}') )
FROM Sales.Articles
```

Namensraum	FreD.Understand.Properties
Assembly	Core
Klasse	MinProperty

Beschreibung Ermittelt die n kleinsten Werte einer Spalte.

Eigenschaften

MaxCount	Anzahl der minimalen Werte, die zurückgegeben werden (Default: 5).

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
             '{FreD.Understand.Properties.MinProperty|Core|MinCount=10}'+
             COALESCE( ProductNumber, '{NULL}') )
FROM Sales.Articles
```

Namensraum	FreD.Understand.Properties
Assembly	Core
Klasse	PrimaryKeyCandidateProperty

Beschreibung Ermittelt, ob eine Spalte als Primärschlüssel geeignet ist.

Eigenschaften

Threshold	Schwellwert eindeutiger Werte, ab dem eine Spalte als Primärschlüssel identifiziert wird (Default: 0.95).

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
          '{FreD.Understand.Properties.PrimaryKeyCandidateProperty|
           Core|Threshold=0.8}'+
          COALESCE( Name, '{NULL}') )
FROM Persons.Customers
```

Namensraum	FreD.Understand.Properties
Assembly	Core
Klasse	UniqueCountProperty

Beschreibung Gibt die Anzahl eindeutiger Werte zurück.

Eigenschaften

keine

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
             '{FreD.Understand.Properties.UniqueCountProperty|Core}'+
             COALESCE( Name, '{NULL}') )
FROM Persons.Customers
```

Namensraum	FreD.Understand.Properties
Assembly	Core
Klasse	UniqueCountProperty

Beschreibung Gibt das relative Verhältnis eindeutiger Werte zur Gesamtanzahl zurück.

Eigenschaften

keine

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
             '{FreD.Understand.Properties.UniquenessProperty|Core}'+
             COALESCE( Name, '{NULL}') )
FROM Persons.Customers
```

Namensraum	FreD.Understand.Properties.Text
Assembly	Core
Klasse	BlankCountProperty

Beschreibung Gibt die Anzahl an Werten, die leer (nicht NULL) sind.

Eigenschaften

keine

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
             '{FreD.Understand.Properties.Text.BlankCountProperty|Core}'+
             COALESCE( Title, '{NULL}') )
FROM Persons.Customers
```

Namensraum	FreD.Understand.Properties.Text
Assembly	Core
Klasse	MaxTextCountProperty

Beschreibung Anzahl der Texte mit der größten Länge.

Eigenschaften

keine

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
           '{FreD.Understand.Properties.Text.MaxTextCountProperty|Core}'+
           COALESCE( Name, '{NULL}') )
FROM Persons.Customers
```

Namensraum	FreD.Understand.Properties.Text
Assembly	Core
Klasse	MaxTextLengthProperty

Beschreibung Größe des längsten Textes.

Eigenschaften

keine

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
          '{FreD.Understand.Properties.Text.MaxTextLengthProperty|Core}'+
          COALESCE( Name, '{NULL}') )
FROM Persons.Customers
```

Namensraum	FreD.Understand.Properties.Text
Assembly	Core
Klasse	MinTextCountProperty

Beschreibung Anzahl der Texte mit der kleinsten Länge.

Eigenschaften

keine

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
         '{FreD.Understand.Properties.Text.MinTextCountProperty|Core}'+
         COALESCE( Name, '{NULL}') )
FROM Persons.Customers
```

Namensraum	FreD.Understand.Properties.Text
Assembly	Core
Klasse	MinTextLengthProperty

Beschreibung Größe des kürzesten Textes.

Eigenschaften

keine

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
          '{FreD.Understand.Properties.Text.MinTextLengthProperty|Core}'+
          COALESCE( Name, '{NULL}') )
FROM Persons.Customers
```

Namensraum	FreD.Understand.Properties.Number
Assembly	Core
Klasse	BenfordsLawProperty

Beschreibung Gibt die Verteilung der einzelnen Ziffern nach Benford's Law und in der übergebenen Spalte aus.

Eigenschaften

TrimmedCount	Anzahl der maximalen und minimalen Werte, die bei der Berechnung nicht berücksichtigt werden sollen.

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
             '{FreD.Understand.Properties.Number.BenfordsLawProperty|
              Core|TrimmedCount=5}'+
            COALESCE( CAST( ListPrice AS NVARCHAR ), '{NULL}') )
FROM Sales.Articles
```

Namensraum	FreD.Understand.Properties.Number
Assembly	Core
Klasse	GeometricMeanProperty

Beschreibung Gibt den geometrischen Mittelwert zurück.

Eigenschaften

TrimmedCount	Anzahl der maximalen und minimalen Werte, die bei der Berechnung nicht berücksichtigt werden sollen.

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
         '{FreD.Understand.Properties.Number.GeometricMeanProperty|Core}'+
         COALESCE( CAST( ListPrice AS NVARCHAR ), '{NULL}') )
FROM Sales.Articles
WHERE Listprice < 100
```

Namensraum	FreD.Understand.Properties.Number
Assembly	Core
Klasse	HarmonicMeanProperty

Beschreibung Gibt den harmonischen Mittelwert zurück.

Eigenschaften

TrimmedCount	Anzahl der maximalen und minimalen Werte, die bei der Berechnung nicht berücksichtigt werden sollen.

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
         '{FreD.Understand.Properties.Number.HarmonicMeanProperty|Core}'+
         COALESCE( CAST( ListPrice AS NVARCHAR ), '{NULL}') )
FROM Sales.Articles
```

Namensraum	FreD.Understand.Properties.Number
Assembly	Core
Klasse	MeanProperty

Beschreibung Gibt den arithmetischen Mittelwert zurück.

Eigenschaften

TrimmedCount	Anzahl der maximalen und minimalen Werte, die bei der Berechnung nicht berücksichtigt werden sollen.

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
             '{FreD.Understand.Properties.Number.MeanProperty|Core}'+
             COALESCE( CAST( ListPrice AS NVARCHAR ), '{NULL}') )
FROM Sales.Articles
```

Namensraum	FreD.Understand.Properties.Number
Assembly	Core
Klasse	ProductProperty

Beschreibung Gibt das Produkt aus allen Zahlen zurück.

Eigenschaften

TrimmedCount	Anzahl der maximalen und minimalen Werte, die bei der Berechnung nicht berücksichtigt werden sollen.

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
             '{FreD.Understand.Properties.Number.ProductProperty|Core}'+
             COALESCE( CAST( ListPrice AS NVARCHAR ), '{NULL}') )
FROM Sales.Articles
WHERE ListPrice < 10
```

Namensraum	FreD.Understand.Properties.Number
Assembly	Core
Klasse	RangeProperty

Beschreibung Gibt den Wertebereich aus allen Zahlen zurück.

Eigenschaften

TrimmedCount	Anzahl der maximalen und minimalen Werte, die bei der Berechnung nicht berücksichtigt werden sollen.

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
             '{FreD.Understand.Properties.Number.RangeProperty|Core}'+
             COALESCE( CAST( StockReorderPoint AS NVARCHAR ), '{NULL}') )
FROM Sales.Articles
WHERE ListPrice < 10
```

Namensraum	FreD.Understand.Properties.Number
Assembly	Core
Klasse	SquareSumProperty

Beschreibung Gibt die Summe der Quadrate zurück.

Eigenschaften

TrimmedCount	Anzahl der maximalen und minimalen Werte, die bei der Berechnung nicht berücksichtigt werden sollen.

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
             '{FreD.Understand.Properties.Number.SquareSumProperty|Core}'+
             COALESCE( CAST( StockReorderPoint AS NVARCHAR ), '{NULL}') )
FROM Sales.Articles
```

Namensraum	FreD.Understand.Properties.Number
Assembly	Core
Klasse	StdDevProperty

Beschreibung Gibt die Standardabweichung zurück.

Eigenschaften

TrimmedCount	Anzahl der maximalen und minimalen Werte, die bei der Berechnung nicht berücksichtigt werden sollen.

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
             '{FreD.Understand.Properties.Number.StdDevProperty|Core}'+
             COALESCE( CAST( ListPrice AS NVARCHAR ), '{NULL}') )
FROM Sales.Articles
```

Namensraum	FreD.Understand.Properties.Number
Assembly	Core
Klasse	StdErrProperty

Beschreibung Gibt den Standardfehler zurück.

Eigenschaften

TrimmedCount	Anzahl der maximalen und minimalen Werte, die bei der Berechnung nicht berücksichtigt werden sollen.

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
             '{FreD.Understand.Properties.Number.StdErrProperty|Core}'+
             COALESCE( CAST( ListPrice AS NVARCHAR ), '{NULL}') )
FROM Sales.Articles
```

Namensraum	FreD.Understand.Properties.Number
Assembly	Core
Klasse	SumProperty

Beschreibung Gibt die Summe zurück.

Eigenschaften

TrimmedCount	Anzahl der maximalen und minimalen Werte, die bei der Berechnung nicht berücksichtigt werden sollen.

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
             '{FreD.Understand.Properties.Number.SumProperty|Core}'+
             COALESCE( CAST( ListPrice AS NVARCHAR ), '{NULL}') )
FROM Sales.Articles
```

Namensraum	FreD.Understand.Properties.Number
Assembly	Core
Klasse	VarianceProperty

Beschreibung Gibt die Varianz zurück.

Eigenschaften

TrimmedCount	Anzahl der maximalen und minimalen Werte, die bei der Berechnung nicht berücksichtigt werden sollen.

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
             '{FreD.Understand.Properties.Number.VarianceProperty|Core}'+
             COALESCE( CAST( ListPrice AS NVARCHAR ), '{NULL}') )
FROM Sales.Articles
```

Namensraum	FreD.Understand.Properties.Number
Assembly	Core
Klasse	IntegerCountProperty

Beschreibung Gibt die Anzahl an Ganzzahlen zurück.

Eigenschaften

keine

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
         '{FreD.Understand.Properties.Number.IntegerCountProperty|Core}'+
         COALESCE( PostalCode, '{NULL}') )
FROM Persons.Customers
```

Namensraum	FreD.Understand.Properties.Number
Assembly	Core
Klasse	MaxDecimalPlacesProperty

Beschreibung Gibt die größte Anzahl an Nachkommastellen zurück.

Eigenschaften

keine

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
             '{FreD.Understand.Properties.Number.MaxDecimalPlacesProperty|
              Core}'+
             COALESCE( CAST( ListPrice AS NVARCHAR ), '{NULL}') )
FROM Sales.Articles
```

Namensraum	FreD.Understand.Properties.Number
Assembly	Core
Klasse	MinDecimalPlacesProperty

Beschreibung Gibt die kleinste Anzahl an Nachkommastellen zurück.

Eigenschaften

keine

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
             '{FreD.Understand.Properties.Number.MinDecimalPlacesProperty|
              Core}'+
             COALESCE( CAST( ListPrice AS NVARCHAR ), '{NULL}') )
FROM Sales.Articles
```

Namensraum	FreD.Understand.Properties
Assembly	Core
Klasse	DataPropertyListGeneric

Beschreibung Ermittelt für Spalten einer Tabelle alle Daten-Eigenschaften aus Klassen einer Assembly, die *IDataProperty* implementieren.

Eigenschaften

keine

Beispiel:

```
SELECT *
FROM   q.Understand.PropertyListData(
          'Sample.Sales.Articles',
          'Name, ListPrice, Color',
          null,
          'FreD.Understand.Properties.DataPropertyListGeneric',
          'Core', null )
ORDER BY TypeName
```

Namensraum	FreD.Understand.Properties
Assembly	Core
Klasse	SchemaPropertyListGeneric

Beschreibung Ermittelt für Spalten einer Tabelle alle Schema-Eigenschaften über ADO.NET.

Eigenschaften

keine

Beispiel:

```
SELECT *
FROM q.Understand.PropertyListSchema(
          'Sample.Sales.Articles',
          'ArticleId, Name, Stock, ListPrice',
          'FreD.Understand.Properties.SchemaPropertyListGeneric',
          'Core', null )
ORDER BY Name
```

A.3.2 Primär- und Fremdschlüssel

Namensraum	FreD.Understand.Dependencies
Assembly	Core
Klasse	PrimaryKeyAnalyse

Beschreibung Ermittelt mögliche Primärschlüssel (max. zwei Attribute).

Eigenschaften

Threshold	Schwellwert, der angibt, wie groß der Anteil eindeutiger Werte für ein Primärschlüssel sein muss (Default: 0,8).

Beispiel:

```
SELECT *
FROM   q.Understand.PrimaryKey(
           'Sample.Sales.Articles',
           '*',
           null,
           'FreD.Understand.Dependencies.PrimaryKeyAnalyse',
           'Core',
           'Threshold=0.5' )
```

Columns	Ratio
ArticleId	1
ProductNumber	1
Name, Color	1
Name, Stock	1
Name, StockReorderPoint	1
Name, Size	1
Name, SizeUnitMeasureCode	1
Name, WeightUnitMeasureCode	1
Name, Weight	1
Name, Description	1
Color, Weight	0,748148148148148
Stock, Weight	0,5
Size, Weight	0,507407407407407

Namensraum	FreD.Understand.Dependencies
Assembly	Core
Klasse	ForeignKeyAnalyse

Beschreibung Ermittelt mögliche Primär-/Fremdschlüsselbeziehungen zwischen zwei Tabellen, indem die Übereinstimmung von Werten zwischen zwei Spalten überprüft wird.

Eigenschaften

Threshold — Schwellwert, der angibt, wie groß der Anteil übereinstimmender Werte für ein Primär-/Fremdschlüsselbeziehung sein muss (Default: 0,0), siehe Spalte „Ratio“.

Beispiel:

```
SELECT *
FROM   q.Understand.ForeignKey(
             'Sample.Sales.Articles', '*', null,
             'Sample.Sales.Orders',   '*', null,
             'FreD.Understand.Dependencies.ForeignKeyAnalyse',
             'Core',
             'Threshold=0' )
```

Relation		Shared	Left	Right	Ratio
Sample.Sales.Articles.ArticleId Sample.Sales.Orders.ArticleNo	<==>	16	254	0	0,059
Sample.Sales.Articles.ArticleId Sample.Sales.Orders.CustomerNo	<==>	20	250	2	0,073
Sample.Sales.Articles.Stock Sample.Sales.Orders.ArticleNo	<==>	0	270	25	0
Sample.Sales.Articles.Stock Sample.Sales.Orders.CustomerNo	<==>	98	172	23	0,33
Sample.Sales.Articles.StockReorderPoint Sample.Sales.Orders.ArticleNo	<==>	0	270	25	0
Sample.Sales.Articles.StockReorderPoint Sample.Sales.Orders.CustomerNo	<==>	0	270	25	0

Namensraum	FreD.Understand.Dependencies
Assembly	Core
Klasse	ForeignKeyTextMatching

Beschreibung Ermittelt mögliche Primär-/Fremdschlüsselbeziehungen zwischen zwei Tabellen, indem die Übereinstimmung anhand der Spaltennamen über ein Vergleichsalgorithmus überprüft wird.

Eigenschaften

Threshold	Schwellwert, der angibt, wie groß die Übereinstimmung der Spaltennamen sein muss (Default: 0,9).
Match	Textvergleichsverfahren zum Vergleichen der Spaltennamen (Default: JaroWinklerLynchApproximateMatching).

Beispiel:

```
SELECT *
FROM   q.Understand.ForeignKey(
         'Sample.Sales.Articles', '*', null,
         'Sample.Sales.Orders',   '*', null,
         'FreD.Understand.Dependencies.ForeignKeyTextMatching',
         'Core',
         'Threshold=0.8,
         {Match=FreD.Text.Matching.JaroApproximateMatching}' )
```

Relation	SharedV	LeftV	RightV	Ratio
Sample.Sales.Articles.ArticleId <==> Sample.Sales.Orders.ArticleNo	NULL	NULL	NULL	0,85

A.3.3 Regelinduktion

Namensraum	FreD.Understand.Rules
Assembly	Core
Klasse	OperatorRuleInduction

Beschreibung Sucht nach Größer-, Kleiner-, Gleich-Regeln in den übergebenen Spalten.

Eigenschaften

MinSupport	Minimaler Support der von einer Regel erfüllt sein muss (Default: 0,8).
MinFrequency	Minimale Anzahl an Datensätzen, für die die Regel gelten muss (Default: 0).

Beispiel:

```
SELECT "Rule", Support, Frequency
FROM   q.Understand.RuleInduction(
             'Sample.Sales.Orders',
             '*',
             null,
             'FreD.Understand.Rules.OperatorRuleInduction',
             'Core',
             'MinSupport=0.9, MinFrequency=10')
```

Rule	Support	Frequency
[OrderDate] < [ShipDate]	0,947	18

Namensraum	FreD.Understand.Rules
Assembly	WekaRuleInduction
Klasse	AprioriRuleInduction

Beschreibung Sucht über den Apriori-Algorithmus nach Regeln in den übergebenen Spalten.

Eigenschaften

MinSupportLowerBound	Unterer Wert für den minimalen Support (Default: 0,1).
MaxSupportLowerBound	Oberer Wert für den minimalen Support (Default: 1,0).
MinItemFrequency	Minimale Anzahl an Datensätzen, für die die Regel gelten muss (Default: 0).
Delta	Wert, um den der Support iterativ verringert wird (Default: 0,05).
NumRules	Maximale Anzahl an generierten Regeln (Default: 10).
AutoPreProcess	Automatisches Diskretisieren numerischer Attribute (Default: true).
WekaOptions	Weka-Optionen, haben Vorrang vor den anderen Eigenschaften.

Beispiel:

```
SELECT "Rule", Confidence, Support, ExpectedConfidence,
       Frequency, FrequencyCondition, FrequencyResult
FROM   q.Understand.RuleInduction(
             'Sample.Persons.Customers',
             'Age, MaritalStatus, State, Sex',
             null,
             'FreD.Understand.Rules.AprioriRuleInduction',
             'WekaRuleInduction',
             'AutoPreProcess=false, MinSupportLowerBound=0.01,
              NumRules=100, MinItemFrequency=10, Delta=0.1')
```

Rule	Confidence	Support	Expected Confidence	Frequency	Frequency Condition	Frequency Result
IF [MaritalStatus] = 'widowed' THEN [Age] = 'senior'	1	0,17	1	18	18	18
IF [Age] = 'young' AND [State] = 'AR' THEN [Sex] = 'female'	1	0,11	1	12	12	12
IF [Age] = 'middle-aged' AND [Sex] = 'male' THEN	1	0,11	1	12	12	12

[MaritalStatus] = 'married'						
IF [MaritalStatus] = 'widowed' AND [Sex] = 'male' THEN [Age] = 'senior'	1	0,10	1	10	10	10
IF [Age] = 'middle-aged' THEN [MaritalStatus] = 'married'	0,93	0,26	0,93	27	29	27
IF [MaritalStatus] = 'single' THEN [Age] = 'young'	0,91	0,20	0,91	21	23	21

Namensraum	FreD.Understand.Rules
Assembly	WekaRuleInduction
Klasse	TertiusRuleInduction

Beschreibung Sucht über den Tertius-Algorithmus nach Regeln in den übergebenen Spalten.

Eigenschaften

AutoPreProcess	Automatisches Diskretisieren numerischer Attribute (Default: true).
WekaOptions	Weka-Optionen, haben Vorrang vor den anderen Eigenschaften.

Beispiel:

```
SELECT "Rule", Indicator
FROM   q.Understand.RuleInduction(
             'Sample.Persons.Customers',
             'Age, MaritalStatus, State, Sex, City, State',
             null,
             'FreD.Understand.Rules.TertiusRuleInduction',
             'WekaRuleInduction',
             'AutoPreProcess=false')
```

Rule	Indicator
IF [Age] = 'young' THEN [MaritalStatus] = 'single' OR [City] = 'Hot Springs National Park' OR [State] = 'PR'	0,706
IF [Age] = 'young' THEN [MaritalStatus] = 'single' OR [State] = 'PR' OR [City] = 'Hot Springs National Park'	0,706
IF [MaritalStatus] = 'married' THEN [Age] = 'middle-aged' OR [State] = 'NJ' OR [City] = 'Culver'	0,702
IF [MaritalStatus] = 'married' THEN [Age] = 'middle-aged' OR [State] = 'NJ' OR [City] = 'Greenleaf'	0,702

A.4 Verbessern

A.4.1 Daten standardisieren, korrigieren, anreichern

Namensraum	FreD.Improve.Prediction
Assembly	Core
Klasse	LookupPredictor

Beschreibung Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über Nachschlagetabellen.

Eigenschaften

ResolveMultipleValues	Liefert den ersten gefundenen Wert zurück (First), den letzten (Last) oder erzeugt eine Ausnahme, wenn mehr als ein Wert in der Nachschlagetabelle gefunden wurde (Default: Error).

Beispiel:

```
EXEC q.Improve.PredictorBuild
         'q.Lookup.[City.US]',
         'PostalCode',
         'City',
         null,
         'Predict_CityUS',
         'FreD.Improve.Prediction.LookupPredictor',
         'Core',
         'ResolveMultipleValues=First'

SELECT PostalCode, City,
       q.Improve.PredictorPredict(
               'Predict_CityUS',
                q.Tools.Param( PostalCode, 1 ) )
FROM   Persons.Customers
WHERE  City <> q.Improve.PredictorPredict(
                'Predict_CityUS',
                q.Tools.Param( PostalCode, 1 ) )
```

```
_____________________________________________
Predict 'City' from table 'q.Lookup.[City.US]' by input fields 'PostalCode'
SQL: SELECT City FROM q.Lookup.[City.US] WHERE PostalCode = @par1
ResolveMultipleValues = First
```

Namensraum	FreD.Improve.Prediction
Assembly	Core
Klasse	ZeroRPredictor

Beschreibung

Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über den 0R-Algorithmus (für numerische wird der errechnete Mittelwert, für nominale Ausgabespalten der Modus als Vorhersagewert verwendet).

Eigenschaften

keine

Beispiel:

```
EXEC q.Improve.PredictorBuild
          'Sample.Persons.Customers',
          'Sex',
          'Sex',
          null,
          'Predict_Sex',
          'FreD.Improve.Prediction.ZeroRPredictor',
          'Core',
          null

SELECT Name, Sex,
       q.Improve.PredictorPredict(
              'Predict_Sex',
              q.Tools.Param( Sex, 1 ) )
FROM   Persons.Customers
```

```
_____________________________________
ZeroR predicts class value: 'female' (Modal)
```

Namensraum	FreD.Improve.Prediction
Assembly	WekaPrediction
Klasse	GenericPredictor

Beschreibung Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über einen beliebigen Klassifizierungs-Algorithmus, der in der Assembly Weka implementiert ist.

Eigenschaften

ClassifierName	Vollständiger Java-Klassenname in der Weka-Bibliothek
WekaOptions	Weka-Optionen, haben Vorrang vor den anderen Eigenschaften.
CrossValidationEvaluation	Führt eine Kreuzvalidierung durch (Default: false).
ConfusionMatrix	Gibt eine Konfusionsmatrix aus (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true.
ClassDetails	Detaillierte Ausgabe für jede Klasse (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true.

Beispiel:

```
EXEC q.Improve.PredictorBuild
          'Sample.Persons.Customers',
          'Sex, MaritalStatus',
          'Age',
          null,
          'Predict_Age',
          'FreD.Improve.Prediction.GenericPredictor',
          'WekaPrediction',
          'ClassifierName=weka.classifiers.trees.Id3'

SELECT Sex, MaritalStatus, Age,
       q.Improve.PredictorPredict(
           'Predict_Age',
           q.Tools.Param( Sex, 0 ) + q.Tools.Param( MaritalStatus, 1 ) )
FROM   Persons.Customers
```

```
Id3

MaritalStatus = single
|  Sex = female: young
|  Sex = male: young
MaritalStatus = married
|  Sex = female: senior
|  Sex = male: middle-aged
```

```
MaritalStatus = divorced
|  Sex = female: young
|  Sex = male: senior
MaritalStatus = widowed: senior
```

Namensraum	FreD.Improve.Prediction
Assembly	WekaPrediction
Klasse	Id3Predictor

Beschreibung Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über den ID3-Algorithmus.

Eigenschaften

ClassifierName	Vollständiger Java-Klassenname in der Weka-Bibliothek (Default: weka.classifiers.trees.Id3).
WekaOptions	Weka-Optionen, haben Vorrang vor den anderen Eigenschaften.
CrossValidationEvaluation	Führt eine Kreuzvalidierung durch (Default: false).
ConfusionMatrix	Gibt eine Konfusionsmatrix aus (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true.
ClassDetails	Detaillierte Ausgabe für jede Klasse (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true.

Beispiel:

```
EXEC q.Improve.PredictorBuild
          'Sample.Persons.Customers',
          'Sex, MaritalStatus',
          'Age',
          null,
          'Predict_Age',
          'FreD.Improve.Prediction.Id3Predictor',
          'WekaPrediction',
          'CrossValidationEvaluation=true'

SELECT Sex, MaritalStatus, Age,
       q.Improve.PredictorPredict(
          'Predict_Age',
          q.Tools.Param( Sex, 0 ) + q.Tools.Param( MaritalStatus, 1 ) )
FROM   Persons.Customers
```

```
Id3


MaritalStatus = single
|  Sex = female: young
|  Sex = male: young
MaritalStatus = married
|  Sex = female: senior
|  Sex = male: middle-aged
MaritalStatus = divorced
|  Sex = female: young
```

```
|  Sex = male: senior
MaritalStatus = widowed: senior


Correctly Classified Instances          71               69.6078 %
Incorrectly Classified Instances        31               30.3922 %
Kappa statistic                          0.5181
Mean absolute error                      0.2038
Root mean squared error                  0.3274
Relative absolute error                 47.2603 %
Root relative squared error             70.5133 %
Total Number of Instances              102


=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  0.483     0.11       0.636     0.483     0.549    middle-aged
  0.778     0.053      0.84      0.778     0.808    young
  0.783     0.339      0.655     0.783     0.713    senior


=== Confusion Matrix ===

  a  b  c   <-- classified as
 14  2 13 |  a = middle-aged
  0 21  6 |  b = young
  8  2 36 |  c = senior
```

Namensraum	FreD.Improve.Prediction
Assembly	WekaPrediction
Klasse	J48Predictor

Beschreibung	Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über den J48-Algorithmus.

Eigenschaften

ClassifierName	Vollständiger Java-Klassenname in der Weka-Bibliothek (Default: weka.classifiers.trees.J48).
WekaOptions	Weka-Optionen, haben Vorrang vor den anderen Eigenschaften.
CrossValidationEvaluation	Führt eine Kreuzvalidierung durch (Default: false).
ConfusionMatrix	Gibt eine Konfusionsmatrix aus (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true.
ClassDetails	Detaillierte Ausgabe für jede Klasse (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true.

Beispiel:

```
EXEC q.Improve.PredictorBuild
          'Sample.Persons.Customers',
          'Sex, MaritalStatus',
          'Age',
          null,
          'Predict_Age',
          'FreD.Improve.Prediction.J48Predictor',
          'WekaPrediction',
          null

SELECT Sex, MaritalStatus, Age,
       q.Improve.PredictorPredict(
         'Predict_Age',
         q.Tools.Param( Sex, 0 ) + q.Tools.Param( MaritalStatus, 1 ) )
FROM   Persons.Customers
```

```
J48 pruned tree
------------------

MaritalStatus = single: young (23.0/2.0)
MaritalStatus = married
|   Sex = female: senior (35.0/18.0)
|   Sex = male: middle-aged (14.0/2.0)
MaritalStatus = divorced: senior (12.0/3.0)
MaritalStatus = widowed: senior (18.0)

Number of Leaves  :   5
```

```
Size of the tree :    7


Correctly Classified Instances          68               66.6667 %
Incorrectly Classified Instances        34               33.3333 %
Kappa statistic                          0.4874
Mean absolute error                      0.2304
Root mean squared error                  0.3566
Relative absolute error                 53.4365 %
Root relative squared error             76.8017 %
Total Number of Instances              102
```

Namensraum	FreD.Improve.Prediction
Assembly	WekaPrediction
Klasse	LogisticPredictor

Beschreibung	Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über ein multinominales logistisches Regressionsmodell.

Eigenschaften

ClassifierName	Vollständiger Java-Klassenname in der Weka-Bibliothek (Default: weka.classifiers.functions.Logistic).
WekaOptions	Weka-Optionen, haben Vorrang vor den anderen Eigenschaften.
CrossValidationEvaluation	Führt eine Kreuzvalidierung durch (Default: false).
ConfusionMatrix	Gibt eine Konfusionsmatrix aus (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true.
ClassDetails	Detaillierte Ausgabe für jede Klasse (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true.

Beispiel:

```
EXEC q.Improve.PredictorBuild
          'Sample.Persons.Customers',
          'Sex, MaritalStatus',
          'Age',
          null,
          'Predict_Age',
          'FreD.Improve.Prediction.LogisticPredictor',
          'WekaPrediction',
          null

SELECT Sex, MaritalStatus, Age,
       q.Improve.PredictorPredict(
         'Predict_Age',
         q.Tools.Param( Sex, 0 ) + q.Tools.Param( MaritalStatus, 1 ) )
FROM   Persons.Customers
```

```
_____________________________________
Logistic Regression with ridge parameter of 1.0E-8
Coefficients...

Variable      Coeff.
       1       1.2128      -0.7487
       2      18.8286      20.0272
       3       4.063        0.2705
       4     -16.423        1.2674
       5     -17.8746     -25.4368
```

```
Intercept     -4.0365    -2.0268


Odds Ratios...
Variable         O.R.
       1       3.3628       0.473
       2 150371511.5031 498523822.1046
       3      58.149        1.3106
       4       0            3.5515
       5       0            0
```

Correctly Classified Instances	70	68.6275 %
Incorrectly Classified Instances	32	31.3725 %
Kappa statistic	0.5196	
Mean absolute error	0.2212	
Root mean squared error	0.3377	
Relative absolute error	51.2925 %	
Root relative squared error	72.7276 %	
Total Number of Instances	102	

Namensraum	FreD.Improve.Prediction
Assembly	WekaPrediction
Klasse	MultiLayerPerceptronPredictor

Beschreibung	Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über ein neuronales Netz.

Eigenschaften

ClassifierName	Vollständiger Java-Klassenname in der Weka-Bibliothek (Default: weka.classifiers.functions.MultilayerPerceptron).
WekaOptions	Weka-Optionen, haben Vorrang vor den anderen Eigenschaften.
CrossValidationEvaluation	Führt eine Kreuzvalidierung durch (Default: false)
ConfusionMatrix	Gibt eine Konfusionsmatrix aus (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true.
ClassDetails	Detaillierte Ausgabe für jede Klasse (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true.

Beispiel:

```
    EXEC q.Improve.PredictorBuild
            'Sample.Persons.Customers',
            'Sex, MaritalStatus',
            'Age',
            null,
            'Predict_Age',
            'FreD.Improve.Prediction.MultilayerPerceptronPredictor',
            'WekaPrediction',
            'CrossValidationEvaluation=true'

  SELECT Sex, MaritalStatus, Age,
         q.Improve.PredictorPredict(
             'Predict_Age',
             q.Tools.Param( Sex, 0 ) + q.Tools.Param( MaritalStatus, 1 ) )
  FROM   Persons.Customers
```

```
_____________________________________
Sigmoid Node 0
    Inputs    Weights
    Threshold    0.99538036675303787
    Node 3    -3.2007710608453914
    Node 4    1.17192812053411
    Node 5    -1.4518174715459062
    Node 6    -3.1970914276910709
Sigmoid Node 1
    Inputs    Weights
    Threshold    -0.034139192468001539
    Node 3    2.6950700652637227
    Node 4    -4.4818098768433048
```

```
    Node 5    -2.3834792005996177
    Node 6    -1.1490721730188707
Sigmoid Node 2
    Inputs    Weights
    Threshold    -3.8708761574854567
    Node 3    -0.19673472763195973
    Node 4    1.8424649387380081
    Node 5    4.0954262877598664
    Node 6    4.2545524189808122
Sigmoid Node 3
    Inputs    Weights
    Threshold    0.66570268951633971
    Attrib Sex    -0.065252494335457822
    Attrib MaritalStatus=single    1.503091439141339
    Attrib MaritalStatus=married    -3.4163039668569417
    Attrib MaritalStatus=divorced    0.38509571293920936
    Attrib MaritalStatus=widowed    0.061354959834904688
Sigmoid Node 4
    Inputs    Weights
    Threshold    -0.77652801943204464
    Attrib Sex    2.7699691891519334
    Attrib MaritalStatus=single    -2.7213815471053597
    Attrib MaritalStatus=married    0.62912868785595344
    Attrib MaritalStatus=divorced    0.42342834209180563
    Attrib MaritalStatus=widowed    3.0251891148744763
Sigmoid Node 5
    Inputs    Weights
    Threshold    0.79669112088210159
    Attrib Sex    -3.0291329611001858
    Attrib MaritalStatus=single    -3.6553549005310257
    Attrib MaritalStatus=married    -0.74091091905337314
    Attrib MaritalStatus=divorced    1.5726571467605397
    Attrib MaritalStatus=widowed    1.24633172228247
Sigmoid Node 6
    Inputs    Weights
    Threshold    -0.75310220400548611
    Attrib Sex    2.4804210909493083
    Attrib MaritalStatus=single    -0.87660437584765893
    Attrib MaritalStatus=married    -1.7076710138669786
    Attrib MaritalStatus=divorced    0.69594415286889555
    Attrib MaritalStatus=widowed    3.39063373437556
Class middle-aged
    Input
    Node 0
Class young
    Input
    Node 1
Class senior
    Input
    Node 2



Correctly Classified Instances          71               69.6078 %
Incorrectly Classified Instances        31               30.3922 %
Kappa statistic                          0.5181
Mean absolute error                      0.2101
Root mean squared error                  0.3296
Relative absolute error                 48.7187 %
Root relative squared error             70.9807 %
Total Number of Instances              102
```

Namensraum	FreD.Improve.Prediction
Assembly	WekaPrediction
Klasse	NaiveBayesPredictor

Beschreibung	Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über einen Naive Bayes Klassifizierer.

Eigenschaften

ClassifierName	Vollständiger Java-Klassenname in der Weka-Bibliothek (Default: weka.classifiers.bayes.NaiveBayes).
WekaOptions	Weka-Optionen, haben Vorrang vor den anderen Eigenschaften.
CrossValidationEvaluation	Führt eine Kreuzvalidierung durch (Default: false).
ConfusionMatrix	Gibt eine Konfusionsmatrix aus (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true.
ClassDetails	Detaillierte Ausgabe für jede Klasse (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true.

Beispiel:

```
EXEC q.Improve.PredictorBuild
          'Sample.Persons.Customers',
          'Sex, MaritalStatus',
          'Age',
          null,
          'Predict_Age',
          'FreD.Improve.Prediction.NaiveBayesPredictor',
          'WekaPrediction',
          'CrossValidationEvaluation=true'

SELECT Sex, MaritalStatus, Age,
       q.Improve.PredictorPredict(
          'Predict_Age',
          q.Tools.Param( Sex, 0 ) + q.Tools.Param( MaritalStatus, 1 ) )
FROM   Persons.Customers
```

```
Naive Bayes Classifier

Class middle-aged: Prior probability = 0.29

Sex:  Discrete Estimator. Counts =  18 13  (Total = 31)
MaritalStatus:  Discrete Estimator. Counts =  3 28 1 1  (Total = 33)


Class young: Prior probability = 0.27
```

```
Sex:  Discrete Estimator. Counts =  22 7  (Total = 29)
MaritalStatus:  Discrete Estimator. Counts =  22 4 4 1  (Total = 31)


Class senior: Prior probability = 0.45

Sex:  Discrete Estimator. Counts =  29 19  (Total = 48)
MaritalStatus:  Discrete Estimator. Counts =  1 20 10 19  (Total = 50)



Correctly Classified Instances          75               73.5294 %
Incorrectly Classified Instances        27               26.4706 %
Kappa statistic                          0.6056
Mean absolute error                      0.2609
Root mean squared error                  0.3475
Relative absolute error                 60.497  %
Root relative squared error             74.8359 %
Total Number of Instances              102
```

Namensraum	FreD.Improve.Prediction
Assembly	WekaPrediction
Klasse	NaiveBayesSimplePredictor

Beschreibung	Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über einen einfachen Naive Bayes Klassifizierer.

Eigenschaften

ClassifierName	Vollständiger Java-Klassenname in der Weka-Bibliothek (Default: weka.classifiers.bayes.NaiveBayesSimple).
WekaOptions	Weka-Optionen, haben Vorrang vor den anderen Eigenschaften.
CrossValidationEvaluation	Führt eine Kreuzvalidierung durch (Default: false).
ConfusionMatrix	Gibt eine Konfusionsmatrix aus (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true.
ClassDetails	Detaillierte Ausgabe für jede Klasse (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true.

Beispiel:

```
EXEC q.Improve.PredictorBuild
          'Sample.Persons.Customers',
          'Sex, MaritalStatus',
          'Age',
          null,
          'Predict_Age',
          'FreD.Improve.Prediction.NaiveBayesSimplePredictor',
          'WekaPrediction',
          'CrossValidationEvaluation=true'

SELECT Sex, MaritalStatus, Age,
       q.Improve.PredictorPredict(
          'Predict_Age',
          q.Tools.Param( Sex, 0 ) + q.Tools.Param( MaritalStatus, 1 ) )
FROM   Persons.Customers
```

```
Naive Bayes (simple)

Class middle-aged: P(C) = 0.28571429

Attribute Sex
female  male
0.58064516     0.41935484

Attribute MaritalStatus
```

```
single  married divorced       widowed
0.09090909     0.84848485     0.03030303     0.03030303


Class young: P(C) = 0.26666667

Attribute Sex
female  male
0.75862069     0.24137931

Attribute MaritalStatus
single  married divorced       widowed
0.70967742     0.12903226     0.12903226     0.03225806


Class senior: P(C) = 0.44761905

Attribute Sex
female  male
0.60416667     0.39583333

Attribute MaritalStatus
single  married divorced       widowed
0.02           0.4            0.2            0.38



Correctly Classified Instances          75               73.5294 %
Incorrectly Classified Instances        27               26.4706 %
Kappa statistic                          0.6056
Mean absolute error                      0.2609
Root mean squared error                  0.3475
Relative absolute error                 60.497  %
Root relative squared error             74.8359 %
Total Number of Instances              102
```

Namensraum	FreD.Improve.Prediction
Assembly	WekaPrediction
Klasse	OneRPredictor

Beschreibung	Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über den 1R-Algorithmus.

Eigenschaften

ClassifierName	Vollständiger Java-Klassenname in der Weka-Bibliothek (Default: weka.classifiers.rules.OneR).
WekaOptions	Weka-Optionen, haben Vorrang vor den anderen Eigenschaften.
CrossValidationEvaluation	Führt eine Kreuzvalidierung durch (Default: false).
ConfusionMatrix	Gibt eine Konfusionsmatrix aus (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true.
ClassDetails	Detaillierte Ausgabe für jede Klasse (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true.

Beispiel:

```
    EXEC q.Improve.PredictorBuild
            'Sample.Persons.Customers',
            'Sex, MaritalStatus',
            'Age',
            null,
            'Predict_Age',
            'FreD.Improve.Prediction.OneRPredictor',
            'WekaPrediction',
            'CrossValidationEvaluation=true'

  SELECT Sex, MaritalStatus, Age,
         q.Improve.PredictorPredict(
             'Predict_Age',
             q.Tools.Param( Sex, 0 ) + q.Tools.Param( MaritalStatus, 1 ) )
  FROM   Persons.Customers
```

```
MaritalStatus:
       single  -> young
       married -> middle-aged
       divorced       -> senior
       widowed -> senior
(75/102 instances correct)


Correctly Classified Instances          75               73.5294 %
```

```
Incorrectly Classified Instances        27               26.4706 %
Kappa statistic                          0.6056
Mean absolute error                      0.1765
Root mean squared error                  0.4201
Relative absolute error                 40.9208 %
Root relative squared error             90.4729 %
Total Number of Instances              102
```

Namensraum	FreD.Improve.Prediction
Assembly	WekaPrediction
Klasse	BaggingPredictor

Beschreibung Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über das Bagging-Verfahren.

Eigenschaften

ClassifierName	Vollständiger Java-Klassenname in der Weka-Bibliothek (Default: weka.classifiers.meta.Bagging).
WekaOptions	Weka-Optionen, haben Vorrang vor den anderen Eigenschaften.
CrossValidationEvaluation	Führt eine Kreuzvalidierung durch (Default: false).
ConfusionMatrix	Gibt eine Konfusionsmatrix aus (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true.
ClassDetails	Detaillierte Ausgabe für jede Klasse (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true.
ClassifierNameIntern	Weka-Klassifizierer, der im Bagging-Verfahren verwendet wird (Default: weka.classifiers.rules.ZeroR).

Beispiel:

```
EXEC q.Improve.PredictorBuild
          'Sample.Persons.Customers',
          'Sex, MaritalStatus',
          'Age',
          null,
          'Predict_Age',
          'FreD.Improve.Prediction.BaggingPredictor',
          'WekaPrediction',
          'ClassifierNameIntern=weka.classifiers.rules.OneR,
           CrossValidationEvaluation=true'

SELECT Sex, MaritalStatus, Age,
       q.Improve.PredictorPredict(
           'Predict_Age',
           q.Tools.Param( Sex, 0 ) + q.Tools.Param( MaritalStatus, 1 ) )
FROM   Persons.Customers
```

```
All the base classifiers:

MaritalStatus:
       single  -> young
       married -> middle-aged
```

```
       divorced       -> senior
       widowed -> senior
(73/102 instances correct)


MaritalStatus:
       single  -> young
       married -> middle-aged
       divorced       -> senior
       widowed -> senior
(73/102 instances correct)


MaritalStatus:
       single  -> young
       married -> middle-aged
       divorced       -> senior
       widowed -> senior
(76/102 instances correct)


...


MaritalStatus:
       single  -> young
       married -> middle-aged
       divorced       -> senior
       widowed -> senior
(72/102 instances correct)




Correctly Classified Instances          75               73.5294 %
Incorrectly Classified Instances        27               26.4706 %
Kappa statistic                          0.6056
Mean absolute error                      0.1882
Root mean squared error                  0.4114
Relative absolute error                 43.6488 %
Root relative squared error             88.6107 %
Total Number of Instances              102
```

Namensraum	FreD.Improve.Prediction
Assembly	WekaPrediction
Klasse	LinearRegressionPredictor

Beschreibung	Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über ein lineares Regressionsmodell. Unterstützt auch nominale Eingabeattribute.

Eigenschaften

ClassifierName	Vollständiger Java-Klassenname in der Weka-Bibliothek (Default: weka.classifiers.functions.LinearRegression).
WekaOptions	Weka-Optionen, haben Vorrang vor den anderen Eigenschaften.
CrossValidationEvaluation	Führt eine Kreuzvalidierung durch (Default: false).
ConfusionMatrix	Gibt eine Konfusionsmatrix aus (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true.
ClassDetails	Detaillierte Ausgabe für jede Klasse (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true.

Beispiel:

```
EXEC q.Improve.PredictorBuild
          'Sample.Sales.Articles',
          'Stock, Size, Weight',
          'ListPrice',
          'Size IS NOT NULL AND Weight IS NOT NULL',
          'Predict_ListPrice',
          'FreD.Improve.Prediction.LinearRegressionPredictor',
          'WekaPrediction',
          'CrossValidationEvaluation=true'

SELECT Stock, Size, Weight, ListPrice,
       q.Improve.PredictorPredict(
           'Predict_ListPrice',
           q.Tools.Param( Stock, 0 ) +
           q.Tools.Param( Size, 0 ) +
           q.Tools.Param( Weight, 1 ) )
FROM   Sales.Articles
```

```
Linear Regression Model

ListPrice =

    665.7298 * Size=58,62,52,60,50,54,44,48,42,46,38,56 +
    170.7948 * Size=62,52,60,50,54,44,48,42,46,38,56 +
   -836.8154 * Size=52,60,50,54,44,48,42,46,38,56 +
    479.9273 * Size=60,50,54,44,48,42,46,38,56 +
    657.5633 * Size=50,54,44,48,42,46,38,56 +
   -397.5503 * Size=54,44,48,42,46,38,56 +
   -739.6495 * Size=44,48,42,46,38,56 +
    408.1198 * Size=48,42,46,38,56 +
   -196.7499 * Size=42,46,38,56 +
    444.6499 * Size=46,38,56 +
   -444.6499 * Size=38,56 +
    454.3599 * Size=56 +
   -211.3698 *
Weight=2.92,3.02,3.10,3.20,3.14,3.16,2.32,2.40,2.48,2.46,2.50,2.73,2.77,2.81,2.85,2.36,20.79,
20.13,19.77,20.42,28.68,27.77,28.42,28.13,27.35,2.18,2.38,2.22,2.34,2.26,3.00,2.96,3.04,29.68
,30.00,28.77,29.90,29.42,29.79,29.13,26.77,26.35,18.77,20.00,19.42,19.13,19.79,19.90,27.13,27
.42,3.08,2.30,25.77,25.35,26.42,26.13,17.77,18.13,17.35,18.68,18.42,27.68,27.90,2.80,2.76,2.6
8,2.84,2.72,2.12,2.24,2.20,2.16,17.13,16.77,17.42,17.79,17.90,15.77,15.35,16.42,16.13,23.77,2
3.35,24.13,25.68,25.42,25.90,25.13,14.77,15.13,15.68,15.79,15.42,21.13,20.77,20.35,21.42,14.4
2,14.13,14.68,15.00,13.77 +
    287.8698 *


...


Weight=14.77,15.13,15.68,15.79,15.42,21.13,20.77,20.35,21.42,14.42,14.13,14.68,15.00,13.77 +
   -408.1198 *
Weight=15.13,15.68,15.79,15.42,21.13,20.77,20.35,21.42,14.42,14.13,14.68,15.00,13.77 +
   -257.61   * Weight=15.68,15.79,15.42,21.13,20.77,20.35,21.42,14.42,14.13,14.68,15.00,13.77
+
    666.0206 * Weight=15.42,21.13,20.77,20.35,21.42,14.42,14.13,14.68,15.00,13.77 +
    943.8492 * Weight=21.13,20.77,20.35,21.42,14.42,14.13,14.68,15.00,13.77 +
   -211.3698 * Weight=20.77,20.35,21.42,14.42,14.13,14.68,15.00,13.77 +
   -196.7499 * Weight=21.42,14.42,14.13,14.68,15.00,13.77 +
    599.1906 * Weight=14.42,14.13,14.68,15.00,13.77 +
   -408.4106 * Weight=14.13,14.68,15.00,13.77 +
   -257.61   * Weight=14.68,15.00,13.77 +
   -170.7948 * Weight=15.00,13.77 +
    836.5246 * Weight=13.77 +
    256.92


Correlation coefficient                  0.8119
Mean absolute error                    241.1559
Root mean squared error                538.4216
Relative absolute error                 33.3774 %
Root relative squared error             58.8974 %
Total Number of Instances              176
```

Namensraum	FreD.Improve.Prediction
Assembly	WekaPrediction
Klasse	M5RulesPredictor

Beschreibung	Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über den M5-Algorithmus.

Eigenschaften

ClassifierName	Vollständiger Java-Klassenname in der Weka-Bibliothek (Default: weka.classifiers.meta.Bagging).
WekaOptions	Weka-Optionen, haben Vorrang vor den anderen Eigenschaften.
CrossValidationEvaluation	Führt eine Kreuzvalidierung durch (Default: false).
ConfusionMatrix	Gibt eine Konfusionsmatrix aus (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true.
ClassDetails	Detaillierte Ausgabe für jede Klasse (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true.

Beispiel:

```
EXEC q.Improve.PredictorBuild
          'Sample.Sales.Articles',
          'Stock, Size, Weight',
          'ListPrice',
          'Size IS NOT NULL AND Weight IS NOT NULL',
          'Predict_ListPrice',
          'FreD.Improve.Prediction.M5RulesPredictor',
          'WekaPrediction',
          'CrossValidationEvaluation=true'

SELECT Stock, Size, Weight, ListPrice,
       q.Improve.PredictorPredict(
                 'Predict_ListPrice',
                 q.Tools.Param( Stock, 0 ) +
                 q.Tools.Param( Size, 0 ) +
                 q.Tools.Param( Weight, 1 ) )
FROM   Sales.Articles
```

```
M5 pruned model rules
(using smoothed linear models) :
Number of Rules : 9

Rule: 1
IF
       Weight=15.77,15.35,16.42,16.13,23.77,23.35,24.13,25.68,25.42,25.90,25.13,14.77,15.13,15
.68,15.79,15.42,21.13,20.77,20.35,21.42,14.42,14.13,14.68,15.00,13.77 > 0.5
THEN
```

```
ListPrice =
       -28.0914 * Size=52,60,50,54,44,48,42,46,38,56
       + 33.8038 * Size=60,50,54,44,48,42,46,38,56
       + 23.7558 * Size=50,54,44,48,42,46,38,56
       - 49.5578 * Size=44,48,42,46,38,56
       + 22.4534 * Size=46,38,56
       + 35.8865 *
Weight=2.36,20.79,20.13,19.77,20.42,28.68,27.77,28.42,28.13,27.35,2.18,2.38,2.22,2.34,2.26,3.0
0,2.96,3.04,29.68,30.00,28.77,29.90,29.42,29.79,29.13,26.77,26.35,18.77,20.00,19.42,19.13,19.7
9,19.90,27.13,27.42,3.08,2.30,25.77,25.35,26.42,26.13,17.77,18.13,17.35,18.68,18.42,27.68,27.9
0,2.80,2.76,2.68,2.84,2.72,2.12,2.24,2.20,2.16,17.13,16.77,17.42,17.79,17.90,15.77,15.35,16.42
,16.13,23.77,23.35,24.13,25.68,25.42,25.90,25.13,14.77,15.13,15.68,15.79,15.42,21.13,20.77,20.
35,21.42,14.42,14.13,14.68,15.00,13.77
       + 42.5333 *


...


Rule: 9

ListPrice =
       -230.19 * Size=62,52,60,50,54,44,48,42,46,38,56
       + 567.41 [4/16.846%]



Correlation coefficient                  0.8181
Mean absolute error                    269.5738
Root mean squared error                525.1147
Relative absolute error                 37.3106 %
Root relative squared error             57.4418 %
Total Number of Instances              176
```

Namensraum	FreD.Improve.Prediction
Assembly	WekaPrediction
Klasse	LinearRegressionPredictor

Beschreibung	Ermittelt auf Basis von Eingabeattributen einer Tabelle den Wert für ein Ausgabeattribut über ein einfaches lineares Regressionsmodell.

Eigenschaften

ClassifierName	Vollständiger Java-Klassenname in der Weka-Bibliothek (Default:weka.classifiers.functions.SimpleLinearRegression).
WekaOptions	Weka-Optionen, haben Vorrang vor den anderen Eigenschaften.
CrossValidationEvaluation	Führt eine Kreuzvalidierung durch (Default: false).
ConfusionMatrix	Gibt eine Konfusionsmatrix aus (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true.
ClassDetails	Detaillierte Ausgabe für jede Klasse (Default: false), kann nur aktiv sein, wenn CrossValidationEvaluation=true.

Beispiel:

```
   EXEC q.Improve.PredictorBuild
           'Sample.Sales.Articles',
           'Stock, StockReorderPoint',
           'ListPrice',
           'Size IS NOT NULL AND Weight IS NOT NULL',
           'Predict_ListPrice',
           'FreD.Improve.Prediction.SimpleLinearRegressionPredictor',
           'WekaPrediction',
           'CrossValidationEvaluation=true'

 SELECT Stock, StockReorderPoint, ListPrice,
        q.Improve.PredictorPredict(
            'Predict_ListPrice',
            q.Tools.Param( Stock, 0 ) +
            q.Tools.Param( StockReorderPoint, 1 ) )
 FROM   Sales.Articles
```

```
Linear regression on Stock

-2.21 * Stock + 1809.03
```

```
Correlation coefficient                  0.4561
Mean absolute error                    681.5161
Root mean squared error                809.9723
Relative absolute error                 94.3259 %
Root relative squared error             88.6021 %
Total Number of Instances              176
```

A.4.2 Duplikaterkennung und -fusion

A.4.2.1 Partitioner

Namensraum	FreD.Improve.Deduplication
Assembly	Core
Klasse	BlockingPartitioner

Beschreibung Fasst Datensätze einer Tabelle mit ein oder mehreren Attributen zu Blöcken zusammen.

Eigenschaften

keine

Beispiel:

```
EXEC q.Improve.DeduplicationBuildPartitioner
          'Sample.Persons.Customers',
          'SUBSTRING( Name, 1, 2 )',
          'Name, Street, StreetNumber, PostalCode, City',
          null,
          'Partitioner_Customers',
          'FreD.Improve.Deduplication.BlockingPartitioner',
          'Core',
          null
```

```
_____________________________________________
Partitioner = FreD.Improve.Deduplication.BlockingPartitioner
Table = Sample.Persons.Customers (Name, Street, StreetNumber, PostalCode, City)
MatchKey = SUBSTRING( Name, 1, 2 )
```

Namensraum	FreD.Improve.Deduplication
Assembly	Core
Klasse	CanopyPartitioner

Beschreibung	Fasst Datensätze einer Tabelle über das Canopy-Verfahren zu Partitionen zusammen.

Eigenschaften

CanopyComparator	Textvergleichsalgorithmus (Default: FreD.Text.Matching.Sift3ApproximateMatching).
ThresholdCanopy	Schwellwert, ab dem zwei Werte beim Textvergleich zu einer Partition zugeordnet werden (Default: 0,5).

Beispiel:

```
EXEC q.Improve.DeduplicationBuildPartitioner
        'Sample.Persons.Customers',
        'Name',
        'Name, Street, StreetNumber, PostalCode, City',
        null,
        'Partitioner_Customers',
        'FreD.Improve.Deduplication.CanopyPartitioner',
        'Core',
  '{CanopyComparator=FreD.Text.Matching.JaroApproximateMatching},
         ThresholdCanopy=0.6'
```

```
_____________________________________________
Number of partitions: 1

101 duplicates in 33 groups detected.
5151 record pairs compared and 5151 record pairs are effectively compared.
Duration: 00:00:00.2343750

Partitioner = FreD.Improve.Deduplication.CanopyPartitioner
Table = Sample.Persons.Customers (Name, Street, StreetNumber, PostalCode, City)
MatchKey = Name
Number of partitions: 1

CanopyComparator = FreD.Text.Matching.JaroApproximateMatching
DuplicateDetector: FreD.Improve.Deduplication.SimpleDetector
GenericComparator: FreD.Text.Matching.JaroApproximateMatching
Threshold = 0,6
Trace = False
CompareAmongEachOther = False
Partitioner = FreD.Improve.Deduplication.CanopyPartitioner
Table = Sample.Persons.Customers (Name, Street, StreetNumber, PostalCode, City)
MatchKey = Name
Number of partitions: 1

CanopyComparator = FreD.Text.Matching.JaroApproximateMatching
```

Namensraum	FreD.Improve.Deduplication
Assembly	Core
Klasse	RulebasedPartitioner

Beschreibung Fasst Datensätze einer Tabelle zu Partitionen zusammen, indem in SQL Regeln beschrieben werden, die für zwei Datensätze „a“ und „b“ übereinstimmen müssen.

Eigenschaften

keine

Beispiel:

```
EXEC q.Improve.DeduplicationBuildPartitioner
        'Sample.Persons.Customers',
        'q.Text.MatchingSift3( a.Name, b.Name) > 0.5 AND
         q.Distance.MatchingSimple( a.PostalCode,
                                    b.PostalCode,
                                   ''US'', 100 ) > 0.5',
        'Name, Street, StreetNumber, PostalCode, City',
        null,
        'Partitioner_Customers',
        'FreD.Improve.Deduplication.RulebasedPartitioner',
        'Core',
        null
```

```
_____________________________________________
Partitioner = FreD.Improve.Deduplication.RulebasedPartitioner
Table = Sample.Persons.Customers (Name, Street, StreetNumber, PostalCode, City)
MatchKey = q.Text.MatchingSift3( a.Name, b.Name) > 0.5 AND
          q.Distance.MatchingSimple( a.PostalCode, b.PostalCode, 'US', 100 ) > 0.5
Rules = q.Text.MatchingSift3( a.Name, b.Name) > 0.5 AND
          q.Distance.MatchingSimple( a.PostalCode, b.PostalCode, 'US', 100 ) > 0.5

31 Duplicate groups detected
```

Namensraum	FreD.Improve.Deduplication
Assembly	Core
Klasse	TokenizerPartitioner

Beschreibung Fasst Datensätze einer Tabelle zu Partitionen zusammen, für die die Token von einem oder mehreren Attributen übereinstimmen.

Eigenschaften

Tokenizer	Tokenizer-Klasse, die zum Aufteilen in Token verwendet werden soll (Default: NGramTokenizer).

Beispiel:

```
EXEC q.Improve.DeduplicationBuildPartitioner
     'Sample.Persons.Customers',
     'Name',
     'Name, Street, StreetNumber, PostalCode, City',
     null,
     'Partitioner_Customers',
     'FreD.Improve.Deduplication.TokenizerPartitioner',
     'Core',
     '{Tokenizer=FreD.Text.Tokenizer.NGramThresholdTokenizer,
     NGramSize=3, Threshold=0.7}'
```

```
_____________________________________________
Partitioner = FreD.Improve.Deduplication.TokenizerPartitioner
Table = Sample.Persons.Customers (Name, Street, StreetNumber, PostalCode, City)
MatchKey = Name


FreD.Text.Tokenizer.NGramThresholdTokenizer
```

Namensraum	FreD.Improve.Deduplication
Assembly	Core
Klasse	WindowingPartitioner

Beschreibung Fasst Datensätze einer Tabelle mit ein oder mehreren Attributen zu einer sortierten Liste zusammen und durchläuft diese nach der Sorted-Neighborhood-Methode.

Eigenschaften

WindowSize	Größe des weiter zu schiebenden Fensters (Default: 10).

Beispiel:

```
EXEC q.Improve.DeduplicationBuildPartitioner
          'Sample.Persons.Customers',
          'SUBSTRING( Name, 1, 2 )',
          'Name, Street, StreetNumber, PostalCode, City',
          null,
          'Partitioner_Customers',
          'FreD.Improve.Deduplication.WindowingPartitioner',
          'Core',
          'WindowSize=10'
```

```
_____________________________________________
Partitioner = FreD.Improve.Deduplication.WindowingPartitioner
Table = Sample.Persons.Customers (Name, Street, StreetNumber, PostalCode, City)
MatchKey = SUBSTRING( Name, 1, 2 )

Window size = 10
```

A.4.2.2 Detector

Namensraum	FreD.Improve.Deduplication
Assembly	Core
Klasse	FellegiSunterDetector

Beschreibung	Vergleicht zwei Datensätze nach dem probabilistischen Fellegi & Sunter-Verfahren.

Eigenschaften

Threshold	Schwellwert, ab dem zwei Datensätze als identisch angesehen werden (Default: 10).
GenericComparator	Vergleichsalgorithmus (default: FreD.Text.Matching.JaroApproximatMatching).
Trace	Gibt die [q.ID] bei Duplikaten und die berechnete Übereinstimmung zurück (Default: false).
ScoreMatch	Wert zwischen 0 und 1, der bestimmt, ab wann zwei Token als identisch angesehen werden (Default: 0,8).
FrequencyTable	Verweis auf Tabelle im Repository mit dem Präfix „frq“ mit den relativen Häufigkeiten einzelner Tokens (kann mit der SQL-Prozedur *Tools.BuildFrequencyTable* erzeugt werden).
MatchProbability	Häufigkeitsverhältnis bei Übereinstimmung zur Gewichtsberechnung (Default: 0,95).
UnmatchProbability	Häufigkeitsverhältnis bei Nicht-Übereinstimmung zur Gewichtsberechnung (Default: 0,01).
MissingWeight	Gewicht, wenn Wert fehlt.

Beispiel:

```
EXEC q.Tools.BuildFrequencyTable
          'PersonsCustomers',
          'Sample.Persons.Customers',
          'Name, Street, StreetNumber, PostalCode, City',
          null, 1, 0
```

```
  EXEC q.Improve.DeduplicationBuildDetector
        'Detector_Customers',
        'FreD.Improve.Deduplication.FellegiSunterDetector',
        'Core',
        'Threshold=10.0,
         ScoreMatch=0.75,
         {GenericComparator=FreD.Text.Matching.Sift3ApproximateMatching},
         Trace=false,
         FrequencyTable=PersonsCustomers'
```

```
_____________________________________________
DuplicateDetector: FreD.Improve.Deduplication.FellegiSunterDetector
GenericComparator: FreD.Text.Matching.Sift3ApproximateMatching
Threshold = 10
Trace = False
FrequencyTable = PersonsCustomers
Normalize = False
```

Namensraum	FreD.Improve.Deduplication
Assembly	Core
Klasse	SoftTFIDFDetector

Beschreibung Vergleicht zwei Datensätze nach dem probabilistischen TF-IDF-Verfahren. Beim SoftTFIDFDetector wird jedes Attribut des einen Datensatzes mit jedem Attribut des anderen Datensatzes verglichen und die Übereinstimmung mit dem höchsten Wert verwendet.

Eigenschaften

Threshold	Schwellwert, ab dem zwei Datensätze als identisch angesehen werden (Default: 0,8).
GenericComparator	Vergleichsalgorithmus (default: FreD.Text.Matching.JaroApproximatMatching)
Trace	Gibt die [q.ID] bei Duplikaten und die berechnete Übereinstimmung zurück (Default: false).
ScoreMatch	Wert zwischen 0 und 1, der bestimmt, ab wann zwei Token als identisch angesehen werden (Default: 0,8).
FrequencyTable	Verweis auf Tabelle im Repository mit dem Präfix „frq“ mit den relativen Häufigkeiten einzelner Tokens (kann mit der SQL-Prozedur *Tools.BuildFrequencyTable* erzeugt werden).

Beispiel:

```
EXEC q.Tools.BuildFrequencyTable
          'PersonsCustomers',
          'Sample.Persons.Customers',
          'Name, Street, StreetNumber, PostalCode, City',
          null, 1, 0

EXEC q.Improve.DeduplicationBuildDetector
          'Detector_Customers',
          'FreD.Improve.Deduplication.SoftTFIDFDetector',
          'Core',
          'Trace=true,
           FrequencyTable=PersonsCustomers'
```

```
_____________________________________________
DuplicateDetector: FreD.Improve.Deduplication.SoftTFIDFDetector
GenericComparator: FreD.Text.Matching.JaroApproximateMatching
Threshold = 0,8
Trace = True
FrequencyTable = PersonsCustomers
ScoreMatch = 0,8
```

Namensraum	FreD.Improve.Deduplication
Assembly	Core
Klasse	TFIDFDetector

Beschreibung Vergleicht zwei Datensätze nach dem probabilistischen TF-IDF-Verfahren. Beim TFIDFDetector wird jedes Attribut des einen Datensatzes mit jedem Attribut des anderen Datensatzes verglichen und die Übereinstimmung mit dem höchsten Wert verwendet.

Eigenschaften

Threshold	Schwellwert, ab dem zwei Datensätze als identisch angesehen werden (Default: 0,8).
GenericComparator	Vergleichsalgorithmus (default: FreD.Text.Matching.JaroApproximatMatching).
Trace	Gibt die [q.ID] bei Duplikaten und die berechnete Übereinstimmung zurück (Default: false).
ScoreMatch	Wert zwischen 0 und 1, der bestimmt, ab wann zwei Token als identisch angesehen werden (Default: 0,8).
FrequencyTable	Verweis auf Tabelle im Repository mit dem Präfix „frq“ mit den relativen Häufigkeiten einzelner Tokens (kann mit der SQL-Prozedur *Tools.BuildFrequencyTable* erzeugt werden).
WeightFormula	Formel zur Berechnung des Gewichtes. tf=Platzhalter für Tokenfrequency, idf=Platzhalter für Inverse Document Frequency. (Default: log(tf + 1) * log(idf)). Beispiele: tf * log(idf) log(tf + 1) * log(idf) log(tf / max(tf)) * log(idf)
DividendFormula	Dividend-Formel zur Berechnung des Gesamtgewichtes. (Default: SUM|w1 * w2). Beispiele: SUM|w1 * w2 * matchScore PRODUCT|log(w1 * w2)
DivisorFormula	Dividend-Formel zur Berechnung des Gesamtgewichtes

(Default: PRODUCT|SQRT|SUM|w^2).

Beispiele:

SUM|w^2

Beispiel:

```
EXEC q.Tools.BuildFrequencyTable
          'PersonsCustomers',
          'Sample.Persons.Customers',
          'Name, Street, StreetNumber, PostalCode, City',
          null, 1, 0

EXEC q.Improve.DeduplicationBuildDetector
          'Detector_Customers',
          'FreD.Improve.Deduplication.TFIDFDetector',
          'Core',
          'Trace=true,
           FrequencyTable=PersonsCustomers,
           WeightFormula=tf * log(idf),
           DividendFormula=SUM|w1 * w2 * matchScore,
           DivisorFormula=PRODUCT|SQRT|SUM|w^2'
```

```
_____________________________________________
DuplicateDetector: FreD.Improve.Deduplication.TFIDFDetector
GenericComparator: FreD.Text.Matching.JaroApproximateMatching
Threshold = 0,8
Trace = True
FrequencyTable = PersonsCustomers
w = tf * log(idf)
Dividend = SUM|w1 * w2 * matchScore
Divisor = PRODUCT|SQRT|SUM|w^2
ScoreMatch = 0,8
```

Namensraum	FreD.Improve.Deduplication
Assembly	Core
Klasse	SimpleDetector

Beschreibung	Vergleicht zwei Datensätze über ein einfaches deterministisches Verfahren, das alle Werte untereinander vergleicht und die maximale Übereinstimmung für jede Spalte summiert und die Gesamtsumme durch die Anzahl der Spalten dividiert.

Eigenschaften

Threshold	Schwellwert, ab dem zwei Datensätze als identisch angesehen werden (Default: 0,8).
GenericComparator	Vergleichsalgorithmus (default: FreD.Text.Matching.JaroApproximatMatching).
Trace	Gibt die [q.ID] bei Duplikaten und die berechnete Übereinstimmung zurück (Default: false).
CompareAmongEachOther	Jedes Attribut des einen Datensatzes wird mit jedem Attribut des anderen Datensatzes verglichen (Default: false).

Beispiel:

```
EXEC q.Improve.DeduplicationBuildDetector
          'Detector_Customers',
          'FreD.Improve.Deduplication.SimpleDetector',
          'Core',
          'Trace=true,
           CompareAmongEachOther=true'
```

```
_____________________________________________
DuplicateDetector: FreD.Improve.Deduplication.SimpleDetector
GenericComparator: FreD.Text.Matching.JaroApproximateMatching
Threshold = 0,8
Trace = True
CompareAmongEachOther = True
```

Namensraum	FreD.Improve.Deduplication
Assembly	Core
Klasse	MatchKeyDetector

Beschreibung	Bildet Duplikate aus den Informationen der Partitioner, indem bei einfachen Partitioniereren wie dem Blocking einfach der Sortierschlüssel als Matchkey verwendet wird. Bei Partitionierern, die pro Datensatz mehrere Token verwenden, wird ein token-basierter Vergleichsalgorithmus verwendet, um die Ähnlichkeit von zwei Datensätzen zu überprüfen.

Eigenschaften

Threshold	Schwellwert, ab dem zwei Datensätze als identisch angesehen werden (Default: 0,8).
TokenComparator	Token-basierter Vergleichsalgorithmus, der die ITokenMatching implementiert. (default: FreD.Text.Matching.JaccardApproximatMatching).
Trace	Gibt die [q.ID] bei Duplikaten und die berechnete Übereinstimmung zurück (Default: false).

Beispiel:

```
EXEC q.Improve.DeduplicationBuildDetector
          'Detector_Customers',
          'FreD.Improve.Deduplication.MatchKeyDetector',
          'Core',
          '{TokenComparator=FreD.Text.Matching.CosineApproximateMatching},
           Threshold=0.7'
```

```
_____________________________________________
DuplicateDetector: FreD.Improve.Deduplication.MatchKeyDetector
TokenComparator: FreD.Text.Matching.CosineApproximateMatching
Threshold = 0,7
Trace = False
```

A.4.2.3 Fusion

Namensraum	FreD.Improve.Deduplication.Fuse
Assembly	Core
Klasse	Concat
Beschreibung	Aggregatfunktion, die mehrere Werte zu einer gemeinsamen Zeichenkette verbindet.
Eigenschaften	keine

Beispiel:

```
SELECT q.Improve.DeduplicationFuse(
            '{FreD.Improve.Deduplication.Fuse.Concat|Core}'+
            COALESCE( Name, '{NULL}') )
FROM   Persons.Customers
WHERE  [q.Group] IS NOT NULL
GROUP BY [q.Group]
```

Namensraum	FreD.Improve.Deduplication.Fuse
Assembly	Core
Klasse	Discard

Beschreibung Aggregatfunktion, die nur dann einen Wert zurückgibt, wenn alle Werte der Gruppe identisch sind.

Eigenschaften

keine

Beispiel:

```
SELECT q.Improve.DeduplicationFuse(
            '{FreD.Improve.Deduplication.Fuse.Discard|Core}'+
            COALESCE( Name, '{NULL}') )
FROM   Persons.Customers
WHERE  [q.Group] IS NOT NULL
GROUP BY [q.Group]
```

Namensraum	FreD.Improve.Deduplication.Fuse
Assembly	Core
Klasse	First

Beschreibung Aggregatfunktion, die den ersten Wert einer Gruppe zurückgibt.

Eigenschaften

keine

Beispiel:

```
SELECT q.Improve.DeduplicationFuse(
            '{FreD.Improve.Deduplication.Fuse.First|Core}'+
            COALESCE( Name, '{NULL}') )
FROM   Persons.Customers
WHERE  [q.Group] IS NOT NULL
GROUP BY [q.Group]
```

Namensraum	FreD.Improve.Deduplication.Fuse
Assembly	Core
Klasse	Last

Beschreibung Aggregatfunktion, die den letzten Wert einer Gruppe zurückgibt.

Eigenschaften

keine

Beispiel:

```
SELECT q.Improve.DeduplicationFuse(
            '{FreD.Improve.Deduplication.Fuse.Last|Core}'+
            COALESCE( Name, '{NULL}') )
FROM   Persons.Customers
WHERE  [q.Group] IS NOT NULL
GROUP BY [q.Group]
```

Namensraum	FreD.Improve.Deduplication.Fuse
Assembly	Core
Klasse	Max

Beschreibung Aggregatfunktion, die den größten Wert einer Gruppe zurückgibt.

Eigenschaften

keine

Beispiel:

```
SELECT q.Improve.DeduplicationFuse(
            '{FreD.Improve.Deduplication.Fuse.Max|Core}'+
            COALESCE( Name, '{NULL}') )
FROM   Persons.Customers
WHERE  [q.Group] IS NOT NULL
GROUP BY [q.Group]
```

Namensraum	FreD.Improve.Deduplication.Fuse
Assembly	Core
Klasse	Min

Beschreibung

Aggregatfunktion, die den kleinsten Wert einer Gruppe zurückgibt.

Eigenschaften

keine

Beispiel:

```
SELECT q.Improve.DeduplicationFuse(
            '{FreD.Improve.Deduplication.Fuse.Min|Core}'+
            COALESCE( Name, '{NULL}') )
FROM   Persons.Customers
WHERE  [q.Group] IS NOT NULL
GROUP BY [q.Group]
```

Namensraum	FreD.Improve.Deduplication.Fuse
Assembly	Core
Klasse	MaxFrequency

Beschreibung

Aggregatfunktion, die den am häufigsten vorkommenden Wert der Gruppe zurückgibt.

Eigenschaften

keine

Beispiel:

```
SELECT q.Improve.DeduplicationFuse(
            '{FreD.Improve.Deduplication.Fuse.MaxFrequency|Core}'+
            COALESCE( Name, '{NULL}') )
FROM   Persons.Customers
WHERE  [q.Group] IS NOT NULL
GROUP BY [q.Group]
```

Namensraum	FreD.Improve.Deduplication.Fuse
Assembly	Core
Klasse	LongestText
Beschreibung	Aggregatfunktion, die die längste Zeichenkette der Gruppe zurückgibt.
Eigenschaften	keine

Beispiel:

```
SELECT q.Improve.DeduplicationFuse(
            '{FreD.Improve.Deduplication.Fuse.LongestText|Core}'+
            COALESCE( Name, '{NULL}') )
FROM   Persons.Customers
WHERE  [q.Group] IS NOT NULL
GROUP BY [q.Group]
```

Namensraum	FreD.Improve.Deduplication.Fuse
Assembly	Core
Klasse	ShortestText

Beschreibung

Aggregatfunktion, die die kürzeste Zeichenkette der Gruppe zurückgibt.

Eigenschaften

keine

Beispiel:

```
SELECT q.Improve.DeduplicationFuse(
            '{FreD.Improve.Deduplication.Fuse.ShortestText|Core}'+
            COALESCE( Name, '{NULL}') )
FROM   Persons.Customers
WHERE  [q.Group] IS NOT NULL
GROUP BY [q.Group]
```

B SQL-Schnittstelle

B.1 Allgemein

Alle Verfahren des Datenqualitäts-Frameworks befinden sich in der Datenbank „q“. Über verschiedene Schemata und Präfixe wird eine hierarchische Abbildung der Verfahren ähnlich der von Namensräumen realisiert.

Die wesentliche Logik des Datenqualitäts-Frameworks ist in der Klassenbibliothek implementiert, deren Verfahren über generische SQL-Routinen unter Angabe des vollständigen Klassennamens oder aber über spezifische SQL-Routinen, die direkt eine gewünschte Klasse instanziieren, umgesetzt ist.

Im Folgenden werden die gespeicherten Prozeduren und Funktionen zur Anwendung des Datenqualitäts-Frameworks für das jeweilige Schema alphabetisch geordnet beschrieben.

B.2 Date

Routine	Date.Match
Typ	Skalarwertfunktion
Beschreibung	Errechnet die Ähnlichkeit zwischen zwei Datumswerten, indem über einen generischen Aufruf eine Instanz der übergebenen Klasse instanziiert wird, die die Schnittstelle *IMatching* implementiert hat.

Parameter

@date1	Erstes zu vergleichendes Datum.
@date2	Zweites zu vergleichendes Datum.
@type	Vollständiger Klassenname.
@assemblyString	Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet.
@namedParameters	Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden.

Beispiel:

```
SELECT OrderDate, ShipDate,
       Date.Match( OrderDate, ShipDate,
       'FreD.Date.Matching.DateMatching',
       'Core',
       'MaxTimeSpan=14.00:00:00' )
FROM Sales.Orders
```

Routine	Date.MatchingSimple
Typ	Skalarwertfunktion
Beschreibung	Errechnet die Ähnlichkeit zwischen zwei Datumswerten, indem eine maximale Zeitspanne als Differenz übergeben wird.

Parameter

@date1	Erstes zu vergleichendes Datum.
@date2	Zweites zu vergleichendes Datum.
@type	Vollständiger Klassenname.
@maxTimeSpan	Zeitspanne, die maximal zwischen den Datumswert liegen soll.

Beispiel:

```
SELECT OrderDate, ShipDate,
       q.Date.MatchingSimple( OrderDate, ShipDate, '14.00:00:00' )
FROM Sales.Orders
```

B.3 Distance

Routine	Distance.Match
Typ	Skalarwertfunktion
Beschreibung	Errechnet die Ähnlichkeit zwischen zwei Postleitzahlen, indem über einen generischen Aufruf eine Instanz der übergebenen Klasse instanziiert wird, die die Schnittstelle *IMatching* implementiert hat.

Parameter

@zipCode1	Erste Postleitzahl.
@zipCode 2	Zweite Postleitzahl.
@type	Vollständiger Klassenname.
@assemblyString	Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet.
@namedParameters	Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden.

Beispiel:

```
SELECT a.PostalCode, b.PostalCode,
       q.Distance.Match( a.PostalCode, b.PostalCode,
                         'FreD.Distance.Matching.GeoDistanceMatching',
                         'Core',
                         'MaxDistance=100, Country=US' )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Distance.Match( a.PostalCode, b.PostalCode,
                         'FreD.Distance.Matching.GeoDistanceMatching',
                         'Core',
                         'MaxDistance=100, Country=US' ) > 0.0 AND
       a.CustomerId > b.CustomerId AND
       a.PostalCode <> b.PostalCode
GROUP BY a.PostalCode, b.PostalCode
```

Routine	Distance.MatchingSimple
Typ	Skalarwertfunktion

Beschreibung	Errechnet die Ähnlichkeit zwischen zwei Postleitzahlen, indem über einen spezifischen Aufruf die Klasse *GeoDistanceMatching* instanziiert wird.

Parameter

@zipCode1	Erste Postleitzahl.
@ zipCode 2	Zweite Postleitzahl.
@country	Land, in dem die Postleitzahlen gelten.
@maxDistance	Maximale Distanz in Kilometern, die zwischen den Postleitzahlen liegen darf.

Beispiel:

```
SELECT a.PostalCode, b.PostalCode,
       q.Distance.MatchingSimple(
              a.PostalCode, b.PostalCode, 'US', 100 )
FROM   Persons.Customers a, Persons.Customers b
WHERE  q.Distance.MatchingSimple(
              a.PostalCode, b.PostalCode, 'US', 100 ) > 0.0 AND
       a.CustomerId > b.CustomerId AND a.PostalCode <> b.PostalCode
GROUP BY a.PostalCode, b.PostalCode
```

B.4 Number

Routine	Number.AddPluginEncoderScript
Typ	Gespeicherte Prozedur
Beschreibung	Registriert ein Plug-in als Encoder für Zahlen beim Datenqualitäts-Framework, das über die generischen Funktionen verwendet werden kann.

Parameter

@script	Quellcode in Java, C# oder VB. Falls NULL, wird ein einfaches Standardskript verwendet.
@assemblyName	Name, unter dem das Plug-in als Assembly im RDBMS gespeichert werden soll.

Beispiel:

```
EXEC q.Number.AddPluginEncoderScript
          '
              string code = "";
                      foreach( char c in number )
                      {
                            if( char.IsDigit( c ) )
                                  code += ''9'';
                            else
                                  code += c;
                      }
                      return code;
          ',
          'Plugin_NumberEncoder'


SELECT q.Number.Encode(
              ListPrice,
              'Quality.Number.Encoder.Encoder',
              'Plugin_NumberEncoder',
              NULL ),
       COUNT(*)
FROM   Sales.Articles
GROUP BY q.Number.Encode(
              ListPrice,
              'Quality.Number.Encoder.Encoder',
              'Plugin_NumberEncoder',
              NULL )
```

Routine	Number.AddPluginMatchingScript
Typ	Gespeicherte Prozedur

Beschreibung Registriert ein Plug-in als Vergleichsfunktion für Zahlen beim Datenqualitäts-Framework, das über die generischen Funktionen verwendet werden kann.

Parameter

@script	Quellcode in Java, C# oder VB. Falls NULL, wird ein einfaches Standardskript verwendet.
@assemblyName	Name, unter dem das Plug-in als Assembly im RDBMS gespeichert werden soll.

Beispiel:

```
EXEC q.Number.AddPluginMatchingScript
        '
              double num1 = Convert.ToDouble( number1 );
              double num2 = Convert.ToDouble( number2 );

              if( Math.Max( num1, num2 ) == 0.0 ) return 0.0;

              return 1.0 - ( Math.Abs( num1 - num2 ) /
                             Math.Max( num1, num2 ) );
        ',
        'Plugin_NumberMatching'


SELECT Stock, StockReorderPoint,
       q.Number.Match(
              Stock, StockReorderPoint,
              'Quality.Number.Matching.Matching',
              'Plugin_NumberMatching',
              NULL )
FROM   Sales.Articles
```

Routine Number.CheckDigitsDecode

Typ Skalarwertfunktion

Beschreibung Überprüft die letzte Ziffer einer übergebenen Ziffernfolge auf ihre Gültigkeit als Prüfziffer (Default: Quersumme mit Modulo 10). Ist die Prüfziffer ungültig, wird eine Ausnahme erzeugt, ansonsten die Ziffernfolge ohne die Prüfziffer zurückgeliefert.

Parameter

@text	Ziffernfolge mit Prüfziffer.

Beispiel:

```
SELECT q.Number.CheckDigitsDecode( '12340' )
SELECT q.Number.CheckDigitsDecode( '12341' )
```

Routine	Number.CheckDigitsEncode
Typ	Skalarwertfunktion

Beschreibung Errechnet eine Prüfziffer für eine übergebene Ziffernfolge (Default: Quersumme mit Modulo 10) und gibt die vollständige Ziffernfolge mit Prüfziffer zurück.

Parameter

@text	Ziffernfolge ohne Prüfziffer.

Beispiel:

```
SELECT DISTINCT ProductNumber,
       q.Number.CheckDigitsEncode( ProductNumber )
FROM   Sales.Articles
```

Routine	Number.CheckDigitsVerhoeffDecode
Typ	Skalarwertfunktion

Beschreibung

Überprüft die letzte Ziffer einer übergebenen Ziffernfolge auf ihre Gültigkeit als Prüfziffer nach dem Verhoeff-Algorithmus. Ist die Prüfziffer ungültig, wird eine Ausnahme erzeugt, ansonsten die Ziffernfolge ohne die Prüfziffer zurückgeliefert.

Parameter

@text	Ziffernfolge mit Prüfziffer.

Beispiel:

```
SELECT q.Number.CheckDigitsVerhoeffDecode( '12342' )
SELECT q.Number.CheckDigitsVerhoeffDecode( '12341' )
```

Routine	Number.CheckDigitsVerhoeffEncode
Typ	Skalarwertfunktion

Beschreibung Errechnet eine Prüfziffer für eine übergebene Ziffernfolge nach dem Verhoeff-Algorithmus und gibt die vollständige Ziffernfolge mit Prüfziffer zurück.

Parameter

@text	Ziffernfolge ohne Prüfziffer.

Beispiel:

```
SELECT DISTINCT ProductNumber,
       q.Number.CheckDigitsVerhoeffEncode( ProductNumber )
FROM   Sales.Articles
```

Routine Number.DecimalPlacesEncode

Typ Skalarwertfunktion

Beschreibung Ermittelt die Anzahl der Nachkommastellen von Fließkommazahlen.

Parameter

@number	Zahl.

Beispiel:

```
SELECT DISTINCT ListPrice, q.Number.DecimalPlacesEncode( ListPrice )
FROM   Sales.Articles
```

Routine	Number.Decode
Typ	Skalarwertfunktion

Beschreibung Generische Funktion zum Verwenden von Klassen, die die Kodierung einer Zahl überprüfen.

Parameter

@text	Zahl.
@type	Vollständiger Klassenname.
@assemblyString	Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet.
@namedParameters	Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden.

Beispiel:

```
SELECT q.Number.Decode(   -- Prüfziffernverfahren ISBN-10
          '123456789X',
          'FreD.Number.Encoder.CheckDigitsEncoder',
          'Core',
          'Weights=10|9|8|7|6|5|4|3|2, Modulo=11, Mappings=10:X' )
```

Routine	Number.Encode
Typ	Skalarwertfunktion

Beschreibung Generische Funktion zum Verwenden von Klassen, die Zahlen in einen Code umwandeln.

Parameter

@text	Zahl
@type	Vollständiger Klassenname.
@assemblyString	Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet.
@namedParameters	Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden.

Beispiel:

```
SELECT DISTINCT ProductNumber,
       q.Number.Encode(   -- Prüfziffernverfahren ISBN-10
            ProductNumber,
            'FreD.Number.Encoder.CheckDigitsEncoder',
            'Core',
            'Weights=10|9|8|7|6|5|4|3|2, Modulo=11, Mappings=10:X' )
FROM   Sales.Articles
```

Routine	Number.Match
Typ	Skalarwertfunktion

Beschreibung Generische Funktion zum Verwenden von Klassen, die zwei Zahlen auf Ähnlichkeit überprüfen und einen Indikator hierfür zurückliefern.

Parameter

@number1	Erste Zahl.
@number2	Zweite Zahl.
@type	Vollständiger Klassenname.
@assemblyString	Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet.
@namedParameters	Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden.

Beispiel:

```
SELECT DISTINCT Stock, StockReorderPoint,
       q.Number.Match(
              Stock, StockReorderPoint,
              'FreD.Number.Matching.SimpleNumberMatching',
              'Core',
              'MaxDistance=100' )
FROM Sales.Articles
```

Routine	Number.MatchingEuclidian
Typ	Skalarwertfunktion
Beschreibung	Berechnet auf Grundlage der Euklidischen Distanz zwischen zwei Zahlen einen Ähnlichkeitswert.

Parameter

@number1	Erste Zahl.
@number2	Zweite Zahl.

Beispiel:

```
SELECT DISTINCT Stock, StockReorderPoint,
       q.Number.MatchingEuclidian( Stock, StockReorderPoint )
FROM Sales.Articles
```

Routine	Number.MatchingSimple
Typ	Skalarwertfunktion
Beschreibung	Berechnet auf Grundlage eines maximalen Distanzwertes (Default: 100) zwischen zwei Zahlen einen Ähnlichkeitswert.

Parameter

@number1	Erste Zahl.
@number2	Zweite Zahl.

Beispiel:

```
SELECT DISTINCT Stock, StockReorderPoint,
       q.Number.MatchingSimple( Stock, StockReorderPoint )
FROM Sales.Articles
```

B.5 Text

Routine	Text.AddPluginHashingScript
Typ	Gespeicherte Prozedur
Beschreibung	Registriert ein Plug-in als Hashing Encoder für Text beim Datenqualitäts-Framework, das über die generischen Funktionen verwendet werden kann.

Parameter

@script	Quellcode in Java, C# oder VB. Falls NULL, wird ein einfaches Standardskript verwendet.
@assemblyName	Name, unter dem das Plug-in als Assembly im RDBMS gespeichert werden soll.

Beispiel:

```
EXEC q.Text.AddPluginHashingScript
          '     string code = "";
                int    sum = 0;

                text = text.ToUpper();

                // Quersumme der ersten 3 Alphazeichen berechnen
                int maxLen = 3;
                for( int i = 0;
                     i < System.Math.Min( text.Length, maxLen ); i++ )
                {
                      if( char.IsLetter( text[ i ] ) )
                            sum += ( int )text[ i ] - ( int )''A'' + 1;
                      else
                            maxLen++;
                }
                // Modulo 99 zur Quersumme berechnen
                return ( ( int )( sum % 99 ) ).ToString();
          ',
          'Plugin_TextHashingEncoder'


SELECT Name,
       q.Text.Encode(
              Name,
              'Quality.Text.Encoder.Encoder',
              'Plugin_TextHashingEncoder',
              NULL )
FROM   Sales.Articles
ORDER BY 2
```

Routine	Text.AddPluginMatchingScript
Typ	Gespeicherte Prozedur

Beschreibung Registriert ein Plug-in als Vergleichsfunktion für Text beim Datenqualitäts-Framework, das über die generischen Funktionen verwendet werden kann.

Parameter

@script	Quellcode in Java, C# oder VB. Falls NULL, wird ein einfaches Standardskript verwendet.
@assemblyName	Name, unter dem das Plug-in als Assembly im RDBMS gespeichert werden soll.

Beispiel:

```
EXEC q.Text.AddPluginMatchingScript
          'return text1.IndexOf( text2 ) >= 0 ? 1.0 : 0.0;',
          'Plugin_TextMatching'


SELECT a.CustomerId, a.Name, b.CustomerId, b.Name
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       a.Name <> b.Name AND
       q.Text.Match(
              a.Name, b.Name,
              'Quality.Text.Matching.Matching',
              'Plugin_TextMatching',
              NULL ) = 1.0
```

Routine	Text.AddPluginPatternScript
Typ	Gespeicherte Prozedur

Beschreibung	Registriert ein Plug-in als Pattern Encoder für Text beim Datenqualitäts-Framework, das über die generischen Funktionen verwendet werden kann.

Parameter

@script	Quellcode in Java, C# oder VB. Falls NULL, wird ein einfaches Standardskript verwendet.
@assemblyName	Name, unter dem das Plug-in als Assembly im RDBMS gespeichert werden soll.

Beispiel:

```
EXEC q.Text.AddPluginPatternScript
      '
          string code = "";

          foreach( char c in text )
          {
                if( char.IsLetter( c ) )
                      code += ''X'';
                else if( char.IsDigit( c ) )
                      code += ''9'';
                else code += char.ToUpper( c );
          }

          return code;
      ',
      'Plugin_TextPatternEncoder'


SELECT q.Text.Encode(
              PostalCode,
              'Quality.Text.Encoder.Encoder',
              'Plugin_TextPatternEncoder',
              NULL ),
       COUNT( * )
FROM   Persons.Customers
GROUP BY q.Text.Encode(
              PostalCode,
              'Quality.Text.Encoder.Encoder',
              'Plugin_TextPatternEncoder',
              NULL )
```

Routine	Text.AddPluginPhoneticScript
Typ	Gespeicherte Prozedur

Beschreibung	Registriert ein Plug-in als Phonetic Encoder für Text beim Datenqualitäts-Framework, das über die generischen Funktionen verwendet werden kann.

Parameter

@script	Quellcode in Java, C# oder VB. Falls NULL, wird ein einfaches Standardskript verwendet.
@assemblyName	Name, unter dem das Plug-in als Assembly im RDBMS gespeichert werden soll.

Beispiel:

```
EXEC q.Text.AddPluginPhoneticScript
          '
                text = text.ToUpper();
                text = text.Replace( "A", "" );
                text = text.Replace( "E", "" );
                text = text.Replace( "I", "" );
                text = text.Replace( "O", "" );
                text = text.Replace( "U", "" );
                text = text.Replace( " ", "" );
                return text.Substring( 0,
                    System.Math.Min( 5, text.Length ) );
      ',
          'Plugin_TextPhoneticEncoder'


SELECT q.Text.Encode(
              "Name",
              'Quality.Text.Encoder.Encoder',
              'Plugin_TextPhoneticEncoder',
              NULL )
FROM   Persons.Customers
ORDER BY 1
```

Routine	Text.AddPluginPreparerScript
Typ	Gespeicherte Prozedur

Beschreibung Registriert ein Plug-in als Preparer Encoder für Text beim Datenqualitäts-Framework, das über die generischen Funktionen verwendet werden kann.

Parameter

@script	Quellcode in Java, C# oder VB. Falls NULL, wird ein einfaches Standardskript verwendet.
@assemblyName	Name, unter dem das Plug-in als Assembly im RDBMS gespeichert werden soll.

Beispiel:

```
EXEC q.Text.AddPluginPreparerScript
          'return text.ToUpper();',
          'Plugin_TextPreparerEncoder'


SELECT q.Text.Encode(
              City,
              'Quality.Text.Encoder.Encoder',
              'Plugin_TextPreparerEncoder',
              NULL )
FROM   Persons.Customers
```

Routine	Text.AddPluginTokenizerScript
Typ	Gespeicherte Prozedur
Beschreibung	Registriert ein Plug-in als Tokenizer für Text beim Datenqualitäts-Framework, das über die generischen Funktionen verwendet werden kann.

Parameter

@script	Quellcode in Java, C# oder VB. Falls NULL, wird ein einfaches Standardskript verwendet.
@assemblyName	Name, unter dem das Plug-in als Assembly im RDBMS gespeichert werden soll.

Beispiel:

```
EXEC q.Text.AddPluginTokenizerScript
          'return text.Split( new char[] { '' '', '';'', '','' },
                      System.StringSplitOptions.RemoveEmptyEntries );',
          'Plugin_TextTokenizer'


SELECT q.Text.Tokenize(
              Name,
              ' | ',
              'Quality.Text.Tokenizer.Tokenizer',
              'Plugin_TextTokenizer',
              NULL )
FROM   Persons.Customers
```

Routine	Text.CodeListEncode
Typ	Skalarwertfunktion

Beschreibung	Sequentielle Verknüpfung mehrere Encoder, die einen gemeinsamen Code erzeugen.

Parameter

@text	Zu kodierender Text.
@assemblyString	Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet.
@typesString	List von vollständigen Klassennamen, die Encoder implementieren.

Beispiel:

```
SELECT Name,
     q.Text.CodeListEncode(
         Name,
         'Core',
         'FreD.Text.Encoder.CharSortPatternEncoder|
          FreD.Text.Encoder.DoubleMetaphonePhoneticEncoder|
          FreD.Text.Encoder.SoundexPhoneticEncoder' )
FROM   Persons.Customers
```

Routine	Text.Encode
Typ	Skalarwertfunktion

Beschreibung Generische Funktion zum Verwenden von Klassen, die Text in einen Code umwandeln.

Parameter

@text	Text.
@type	Vollständiger Klassenname.
@assemblyString	Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet.
@namedParameters	Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden.

Beispiel:

```
SELECT Name, q.Text.Encode(
              Name,
              'FreD.Text.Encoder.CharSortPatternEncoder',
              'Core', 'IgnoreMultipleChar=true' )
FROM   Persons.Customers
ORDER BY 2
```

Routine Text.HashingAdler32

Typ Skalarwertfunktion

Beschreibung Berechnet für den übergebenen Wert den Hashwert nach dem Adler32-Algorithmus.

Parameter

@text	Text.

Beispiel:

```
SELECT Name, q.Text.HashingAdler32( Name )
FROM   Persons.Customers
ORDER BY 2
```

Routine	Text.HashingSum
Typ	Skalarwertfunktion
Beschreibung	Berechnet für den übergebenen Wert den Hashwert, indem die Quersumme gebildet wird.

Parameter

@text	Text.

Beispiel:

```
SELECT Name, q.Text.HashingSum( Name )
FROM   Persons.Customers
ORDER BY 2
```

Routine	Text.Match
Typ	Skalarwertfunktion

Beschreibung	Generische Funktion zum Verwenden von Klassen, die zwei Texte auf Ähnlichkeit überprüfen und einen Indikator hierfür zurückliefern.

Parameter

@text1	Erster Text.
@text2	Zweiter Text.
@type	Vollständiger Klassenname.
@assemblyString	Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet.
@namedParameters	Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden.

Beispiel:

```
SELECT a.Name, b.Name,
       q.Text.Match(
             a.Name, b.Name,
             'FreD.Text.Matching.CompressApproximateMatching',
             'CompressApproximateMatching',
             'DeflateAlgorithm=false' )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId
ORDER BY 3 DESC
```

Routine Text.MatchingExact

Typ Skalarwertfunktion

Beschreibung Überprüft zwei Texte auf Gleichheit und liefert bei Übereinstimmung einen Wert von 1, sonst 0 zurück.

Parameter

@text1	Erster Text.
@text2	Zweiter Text.

Beispiel:

```
SELECT a.Name, b.Name
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingExact( a.Name, b.Name ) > 0
```

Routine	Text.MatchingHamming
Typ	Skalarwertfunktion

Beschreibung Berechnet über die Hamming-Distanz zeichenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1	Erster Text.
@text2	Zweiter Text.

Beispiel:

```
SELECT a.Name, b.Name, q.Text.MatchingHamming( a.Name, b.Name )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingHamming( a.Name, b.Name ) > 0.8
ORDER BY 3
```

Routine	Text.MatchingHirschberg
Typ	Skalarwertfunktion

Beschreibung Berechnet über den Hirschberg-Algorithmus zeichenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1	Erster Text.
@text2	Zweiter Text.

Beispiel:

```
SELECT a.Name, b.Name, q.Text.MatchingHirschberg( a.Name, b.Name )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingHirschberg( a.Name, b.Name ) > 0.8
ORDER BY 3
```

Routine	Text.MatchingJaro
Typ	Skalarwertfunktion

Beschreibung Berechnet über den Jaro-Algorithmus zeichenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1	Erster Text.
@text2	Zweiter Text.

Beispiel:

```
SELECT a.Name, b.Name, q.Text.MatchingJaro( a.Name, b.Name )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingJaro( a.Name, b.Name ) > 0.8
ORDER BY 3
```

Routine	Text.MatchingJaroWinkler
Typ	Skalarwertfunktion

Beschreibung Berechnet über den Jaro-Winkler-Algorithmus zeichenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1	Erster Text.
@text2	Zweiter Text.

Beispiel:

```
SELECT a.Name, b.Name, q.Text.MatchingJaroWinkler( a.Name, b.Name )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingJaroWinkler( a.Name, b.Name ) > 0.8
ORDER BY 3
```

Routine Text.MatchingJaroWinklerLynch

Typ Skalarwertfunktion

Beschreibung Berechnet über den Jaro-Winkler-Lynch-Algorithmus zeichenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1	Erster Text.
@text2	Zweiter Text.

Beispiel:

```
SELECT a.Name, b.Name, q.Text.MatchingJaroWinklerLynch( a.Name, b.Name )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingJaroWinklerLynch( a.Name, b.Name ) > 0.8
ORDER BY 3
```

Routine	Text.MatchingLevenshtein
Typ	Skalarwertfunktion

Beschreibung Berechnet über die Damerau-Levenshtein-Distanz zeichenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1	Erster Text.
@text2	Zweiter Text.

Beispiel:

```
SELECT a.Name, b.Name, q.Text.MatchingLevenshtein( a.Name, b.Name )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingLevenshtein( a.Name, b.Name ) > 0.8
ORDER BY 3
```

Routine Text.MatchingLevenshteinHjelmqvist

Typ Skalarwertfunktion

Beschreibung Berechnet über die Damerau-Levenshtein-Distanz zeichenbasiert die Ähnlichkeit zwischen zwei Zeichenketten über den Algorithmus von S. Hjelmqvist und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1	Erster Text.
@text2	Zweiter Text.

Beispiel:

```
SELECT a.Name, b.Name,
       q.Text.MatchingLevenshteinHjelmqvist( a.Name, b.Name )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingLevenshteinHjelmqvist ( a.Name, b.Name ) > 0.8
ORDER BY 3
```

Routine	Text.MatchingRatcliffObershelp
Typ	Skalarwertfunktion

Beschreibung	Berechnet über den Ratcliff-Obershelp-Algorithmus zeichenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1	Erster Text.
@text2	Zweiter Text.

Beispiel:

```
SELECT a.Name, b.Name,
       q.Text.MatchingRatcliffObershelp( a.Name, b.Name )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingRatcliffObershelp ( a.Name, b.Name ) > 0.8
ORDER BY 3
```

Routine Text.MatchingSift3

Typ Skalarwertfunktion

Beschreibung Berechnet zeichenbasiert über den „Longest Common Substring“ die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1	Erster Text.
@text2	Zweiter Text.

Beispiel:

```
SELECT a.Name, b.Name,
       q.Text.MatchingSift3( a.Name, b.Name )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingSift3( a.Name, b.Name ) > 0.8
ORDER BY 3
```

Routine	Text.MatchingTokenCityBlock
Typ	Skalarwertfunktion

Beschreibung Berechnet über die City-Block-Distanz tokenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1	Erster Text.
@text2	Zweiter Text.

Beispiel:

```
SELECT a.Name + ' ' + a.City, b.Name + ' ' + b.City,
       q.Text.MatchingTokenCityBlock( a.Name + ' ' + a.City,
                                      b.Name + ' ' + b.City )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingTokenCityBlock( a.Name + ' ' + a.City,
                                      b.Name + ' ' + b.City ) > 0.5
ORDER BY 3
```

Routine	Text.MatchingTokenCosine
Typ	Skalarwertfunktion

Beschreibung Berechnet über das Kosinus-Maß tokenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1	Erster Text.
@text2	Zweiter Text.

Beispiel:

```
SELECT a.Name + ' ' + a.City, b.Name + ' ' + b.City,
       q.Text.MatchingTokenCosine( a.Name + ' ' + a.City,
                                   b.Name + ' ' + b.City )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingTokenCosine( a.Name + ' ' + a.City,
                                   b.Name + ' ' + b.City ) > 0.5
ORDER BY 3
```

Routine	Text.MatchingTokenDice
Typ	Skalarwertfunktion

Beschreibung Berechnet über das Dice-Maß tokenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1	Erster Text.
@text2	Zweiter Text.

Beispiel:

```
SELECT a.Name + ' ' + a.City, b.Name + ' ' + b.City,
       q.Text.MatchingTokenDice( a.Name + ' ' + a.City,
                                 b.Name + ' ' + b.City )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingTokenDice( a.Name + ' ' + a.City,
                                 b.Name + ' ' + b.City ) > 0.5
ORDER BY 3
```

Routine Text.MatchingTokenEuclidian

Typ Skalarwertfunktion

Beschreibung Berechnet über die Euklidische Distanz tokenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1	Erster Text.
@text2	Zweiter Text.

Beispiel:

```
SELECT a.Name + ' ' + a.City, b.Name + ' ' + b.City,
       q.Text.MatchingTokenEuclidian( a.Name + ' ' + a.City,
                                      b.Name + ' ' + b.City )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingTokenEuclidian( a.Name + ' ' + a.City,
                                      b.Name + ' ' + b.City ) > 0.5
ORDER BY 3
```

Routine Text.MatchingTokenFellegiSunter

Typ Skalarwertfunktion

Beschreibung Berechnet über den Fellegi & Sunter-Algorithmus tokenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1	Erster Text.
@text2	Zweiter Text.

Beispiel:

```
SELECT a.Name + ' ' + a.City, b.Name + ' ' + b.City,
       q.Text.MatchingTokenFellegiSunter( a.Name + ' ' + a.City,
                                          b.Name + ' ' + b.City )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingTokenFellegiSunter( a.Name + ' ' + a.City,
                                          b.Name + ' ' + b.City ) > 8
ORDER BY 3
```

Routine	Text.MatchingTokenJaccard
Typ	Skalarwertfunktion

Beschreibung Berechnet über das Jaccard-Maß tokenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1	Erster Text.
@text2	Zweiter Text.

Beispiel:

```
SELECT a.Name + ' ' + a.City, b.Name + ' ' + b.City,
       q.Text.MatchingTokenJaccard( a.Name + ' ' + a.City,
                                    b.Name + ' ' + b.City )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingTokenJaccard( a.Name + ' ' + a.City,
                                    b.Name + ' ' + b.City ) > 0.5
ORDER BY 3
```

Routine	Text.MatchingTokenMatchingCoefficient
Typ	Skalarwertfunktion

Beschreibung Berechnet über das Matching Coefficient Maß tokenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1	Erster Text.
@text2	Zweiter Text.

Beispiel:

```
SELECT a.Name + ' ' + a.City, b.Name + ' ' + b.City,
       q.Text.MatchingTokenMatchingCoefficient( a.Name + ' ' + a.City,
                                                b.Name + ' ' + b.City )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingTokenMatchingCoefficient( a.Name + ' ' + a.City,
                                                b.Name + ' ' + b.City )
       > 0.5
ORDER BY 3
```

Routine	Text.MatchingTokenOverlap
Typ	Skalarwertfunktion

Beschreibung Berechnet über das Overlap-Maß tokenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1	Erster Text.
@text2	Zweiter Text.

Beispiel:

```
SELECT a.Name + ' ' + a.City, b.Name + ' ' + b.City,
       q.Text.MatchingTokenOverlap( a.Name + ' ' + a.City,
                                    b.Name + ' ' + b.City )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingTokenOverlap( a.Name + ' ' + a.City,
                                    b.Name + ' ' + b.City ) > 0.5
ORDER BY 3
```

Routine	Text.MatchingTokenTFIDF
Typ	Skalarwertfunktion

Beschreibung Berechnet über den TF-IDF-Algorithmus tokenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1	Erster Text.
@text2	Zweiter Text.

Beispiel:

```
SELECT a.Name + ' ' + a.City, b.Name + ' ' + b.City,
       q.Text.MatchingTokenTFIDF( a.Name + ' ' + a.City,
                                  b.Name + ' ' + b.City )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingTokenTFIDF( a.Name + ' ' + a.City,
                                  b.Name + ' ' + b.City ) > 0.5
ORDER BY 3
```

Routine	Text.MatchingTypewriter
Typ	Skalarwertfunktion

Beschreibung Berechnet über die Typewriter-Distanz zeichenbasiert die Ähnlichkeit zwischen zwei Zeichenketten und liefert einen Wert zwischen 0 und 1 zurück.

Parameter

@text1	Erster Text.
@text2	Zweiter Text.

Beispiel:

```
SELECT a.Name, b.Name,
       q.Text.MatchingTypewriter( a.Name, b.Name )
FROM   Persons.Customers a, Persons.Customers b
WHERE  a.CustomerId > b.CustomerId AND
       q.Text.MatchingTypewriter( a.Name, b.Name ) > 0.8
ORDER BY 3
```

Routine Text.PatternsCharSort

Typ Skalarwertfunktion

Beschreibung Sortiert die Zeichen der übergebenen Zeichenkette alphabetisch und liefert diese als Zeichenkette zurück.

Parameter

@text	Text.

Beispiel:

```
SELECT Name, q.Text.PatternsCharSort( Name )
FROM   Persons.Customers
ORDER BY 2
```

Routine Text.PatternsDataTypeRecognised

Typ Skalarwertfunktion

Beschreibung Ermittelt den Datentyp eines Wertes.

Parameter

@text	Text.

Beispiel:

```
SELECT DISTINCT q.Text.PatternsDataTypeRecognised( PostalCode )
FROM   Persons.Customers
```

Routine Text.PatternsNamedEntity

Typ Skalarwertfunktion

Beschreibung Ermittelt den oder die semantischen Typen eines Wertes.

Parameter

@text	Text.

Beispiel:

```
SELECT Name, q.Text.PatternsNamedEntity( Name )
FROM   Persons.Customers
WHERE  q.Text.PatternsNamedEntity( Name ) <> '{FirstName} {LastName}'
```

Routine Text.PatternsOmissionKey

Typ Skalarwertfunktion

Beschreibung Liefert einen Code nach dem Omission Key-Algorithmus von J. Pollock und A. Zamora zurück.

Parameter

@text Text.

Beispiel:

```
SELECT Name, q.Text.PatternsOmissionKey( Name )
FROM   Persons.Customers
ORDER BY 2
```

Routine Text.PatternsReverseOrder

Typ Skalarwertfunktion

Beschreibung Gibt die übergebene Zeichenfolge in umgekehrter Reihenfolge zurück.

Parameter

@text	Text.

Beispiel:

```
SELECT Name, q.Text.PatternsReverseOrder( Name )
FROM   Persons.Customers
ORDER BY 2
```

Routine	Text.PatternsSimple
Typ	Skalarwertfunktion
Beschreibung	Ersetzt Zahlen durch die ‚9', Kleinbuchstaben durch ‚a' und Großbuchstaben durch ‚A'. Alle anderen Zeichen werden durch ein ‚*' ersetzt.

Parameter

@text	Text.

Beispiel:

```
SELECT DISTINCT q.Text.PatternsSimple( PostalCode )
FROM   Persons.Customers
```

Routine Text.PatternsSkeletonKey

Typ Skalarwertfunktion

Beschreibung Liefert einen Code nach dem Skeleton Key-Algorithmus von J. Pollock und A. Zamora zurück.

Parameter

@text	Text.

Beispiel:

```
SELECT Name, q.Text.PatternsSkeletonKey( Name )
FROM   Persons.Customers
ORDER BY 2
```

Routine Text.PhoneticCaver

Typ Skalarwertfunktion

Beschreibung Liefert den phonetischen Caverphone-Code zurück.

Parameter

@text	Text.

Beispiel:

```
SELECT Name, q.Text.PhoneticCaver( Name )
FROM   Persons.Customers
ORDER BY 2
```

Routine Text.PhoneticCologne

Typ Skalarwertfunktion

Beschreibung Liefert den phonetischen Kölner Phonetik-Code zurück.

Parameter

@text	Text.

Beispiel:

```
SELECT Name, q.Text.PhoneticCologne( Name )
FROM   Persons.Customers
ORDER BY 2
```

Routine Text.PhoneticDaitchMokotoff

Typ Skalarwertfunktion

Beschreibung Liefert den phonetischen Code nach Daitch/Mokotoff zurück.

Parameter

@text	Text.

Beispiel:

```
SELECT Name, q.Text.PhoneticDaitchMokotoff( Name )
FROM   Persons.Customers
ORDER BY 2
```

Routine Text.PhoneticDoubleMetaphone

Typ Skalarwertfunktion

Beschreibung Liefert den phonetischen Metaphone-Code zurück.

Parameter

@text	Text.

Beispiel:

```
SELECT Name, q.Text.PhoneticDoubleMetaphone( Name )
FROM   Persons.Customers
ORDER BY 2
```

Routine Text.PhoneticGerman

Typ Skalarwertfunktion

Beschreibung Liefert den phonetischen Code für deutschsprachige Namen nach dem Algorithmus von J. Michael zurück.

Parameter

@text Text..

Beispiel:

```
SELECT Name, q.Text.PhoneticGerman( Name )
FROM   Persons.Customers
ORDER BY 2
```

Routine Text.PhoneticIBMAlphaSearch

Typ Skalarwertfunktion

Beschreibung Liefert den phonetischen Code nach dem „IBM Alpha Inquiry System Personal Name Encoding“-Algorithmus zurück.

Parameter

@text	Text.

Beispiel:

```
SELECT Name, q.Text.PhoneticIBMAlphaSearch( Name )
FROM   Persons.Customers
ORDER BY 2
```

Routine Text.PhoneticNysiis

Typ Skalarwertfunktion

Beschreibung Liefert den phonetischen Code nach dem NYSIIS-Algorithmus („New York State Identification and Intelligence System“) zurück.

Parameter

@text	Text.

Beispiel:

```
SELECT Name, q.Text.PhoneticNysiis( Name )
FROM   Persons.Customers
ORDER BY 2
```

Routine Text.PhoneticONCA

Typ Skalarwertfunktion

Beschreibung Liefert den phonetischen Code nach dem ONCA („Oxford Name Compression Algorithm“) zurück.

Parameter

@text	Text.

Beispiel:

```
SELECT Name, q.Text.PhoneticONCA( Name )
FROM   Persons.Customers
ORDER BY 2
```

Routine Text.PhoneticPhonem

Typ Skalarwertfunktion

Beschreibung Liefert den phonetischen Code nach dem PHONEM-Algorithmus von G. Wilde und C. Meyer zurück.

Parameter

@text	Text.

Beispiel:

```
SELECT Name, q.Text.PhoneticPhonem( Name )
FROM   Persons.Customers
ORDER BY 2
```

Routine Text.PhoneticPhonex

Typ Skalarwertfunktion

Beschreibung Liefert den phonetischen Code nach dem Phonex-Algorithmus von A.J. Lait und B. Randell zurück.

Parameter

@text	Text.

Beispiel:

```
SELECT Name, q.Text.PhoneticPhonex( Name )
FROM   Persons.Customers
ORDER BY 2
```

Routine Text.PhoneticRethSchek

Typ Skalarwertfunktion

Beschreibung Liefert den phonetischen Code für deutsche Namen nach einem Algorithmus von H.-P. von Reth und H.-J. Schek zurück.

Parameter

@text	Text.

Beispiel:

```
SELECT Name, q.Text.PhoneticRethSchek( Name )
FROM   Persons.Customers
ORDER BY 2
```

Routine Text.PhoneticSoundex

Typ Skalarwertfunktion

Beschreibung Liefert den phonetischen Code nach dem Soundex-Algorithmus zurück.

Parameter

@text	Text.

Beispiel:

```
SELECT Name, q.Text.PhoneticSoundex( Name )
FROM   Persons.Customers
ORDER BY 2
```

Routine Text.PreparerGenericText

Typ Skalarwertfunktion

Beschreibung Bereitet den Text für andere Algorithmen vor, indem nicht-druckbare Zeichen entfernt werden.

Parameter

@text	Text.

Beispiel:

```
SELECT DISTINCT Description,
       q.Text.PreparerGenericText( Description )
FROM   Sales.Articles
```

Routine	Text.PreparerRemoveStopwords
Typ	Skalarwertfunktion
Beschreibung	Bereitet den Text für andere Algorithmen vor, indem Stoppwörter entfernt werden.

Parameter

@text	Text.

Beispiel:

```
SELECT DISTINCT Description,
       q.Text.PreparerRemoveStopwords( Description )
FROM   Sales.Articles
```

Routine	Text.Tokenize
Typ	Skalarwertfunktion

Beschreibung Generische Funktion zum Auftrennen von Zeichenketten in einzelne Token durch beliebige Tokenizer-Klassen.

Parameter

@text	Text.
@delimiter	Zeichenkette zur Anzeige der einzelnen getrennten Token
@type	Vollständiger Klassenname.
@assemblyString	Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet.
@namedParameters	Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden.

Beispiel:

```
SELECT Name,
       q.Text.Tokenize( Name, ' | ',
                        'FreD.Text.Tokenizer.NGramTokenizer',
                        'Core',
                        'NGramSize=4, NormalizeHeadTail=false' )
FROM   Persons.Customers
```

Routine	Text.TokenizeFrequency
Typ	Tabellenwertfunktion

Beschreibung Generische Funktion zum Auftrennen von Zeichenketten in einzelne Token durch eine beliebige Tokenizer-Klasse und Berechnen der Häufigkeiten dieser.

Parameter

@table	Tabellenname.
@columns	Spaltennamen.
@where	WHERE-Klausel zur Einschränkung der Werte.
@type	Vollständiger Klassenname.
@assemblyString	Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet.
@namedParameters	Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden.
@distinct	Wenn true, wird ein in einem Datensatz mehrfach vorkommender Wert nur einmal gezählt.

Beispiel:

```
SELECT *
FROM   q.Text.TokenizeFrequency(
                        'Sample.Persons.Customers',
                        'PostalCode',
                        '',
                        'FreD.Text.Tokenizer.NGramTokenizer',
                        'Core',
                        'NGramSize=3, NormalizeHeadTail=false',
                        1 )
 ORDER BY Frequency DESC
```

Routine	Text.TokenizerGeneric
Typ	Skalarwertfunktion

Beschreibung Funktion zum Auftrennen von Zeichenketten in einzelne Token durch die *GenericTokenizer*-Klasse.

Parameter

@text	Text.
@delimiter	Zeichenkette zur Anzeige der einzelnen getrennten Token.

Beispiel:

```
SELECT Name,
       q.Text.TokenizerGeneric( Name, ' | ' )
FROM   Persons.Customers
```

Routine	Text.TokenizerGenericFrequency
Typ	Tabellenwertfunktion

Beschreibung Funktion zum Auftrennen von Zeichenketten in einzelne Token durch die *GenericTokenizer*-Klasse und Berechnen der Häufigkeiten dieser.

Parameter

@table	Tabellenname.
@columns	Spaltennamen.
@where	WHERE-Klausel zur Einschränkung der Werte.
@distinct	Wenn true, wird ein in einem Datensatz mehrfach vorkommender Wert nur einmal gezählt.
@useTokenizer	Wenn true, Verwendung der *GenericTokenizer*-Klasse, ansonsten kein Auftrennen.

Beispiel:

```
SELECT *
FROM   q.Text.TokenizerGenericFrequency(
             'Sample.Persons.Customers', 'Name', NULL, 1, 1 )
ORDER BY Frequency DESC
```

Routine	Text.TokenizerNGram
Typ	Skalarwertfunktion
Beschreibung	Funktion zum Auftrennen von Zeichenketten in einzelne Token durch die *NGramTokenizer*-Klasse.

Parameter

@text	Text.
@delimiter	Zeichenkette zur Anzeige der einzelnen getrennten Token.

Beispiel:

```
SELECT Name,
       q.Text.TokenizerNGram( Name, ' | ' )
FROM   Persons.Customers
```

Routine	Text.TokenizerNGramFrequency
Typ	Tabellenwertfunktion

Beschreibung Funktion zum Auftrennen von Zeichenketten in einzelne Token durch die *NGramTokenizer*-Klasse und Berechnen der Häufigkeiten dieser.

Parameter

@table	Tabellenname.
@columns	Spaltennamen.
@where	WHERE-Klausel zur Einschränkung der Werte.
@distinct	Wenn true, wird ein in einem Datensatz mehrfach vorkommender Wert nur einmal gezählt.
@useTokenizer	Wenn true, Verwendung der *GenericTokenizer*-Klasse, ansonsten kein Auftrennen.

Beispiel:

```
SELECT *
FROM   q.Text.TokenizerNGramFrequency(
            'Sample.Persons.Customers', 'PostalCode', NULL, 1 )
ORDER BY Frequency DESC
```

B.6 Rules

Routine	Rules.AggregateIndicator
Typ	Tabellenwertfunktion
Beschreibung	Liefert für mehrere Regeln einen aggregierten Gesamtindikator für verschiedene Zeiträume zur Beurteilung der Datenqualität zurück.

Parameter

@Where	Einschränkung der Regelmenge.
@startDate	Zeitliche Einschränkung zur Berechnung (Startdatum).
@endDate	Zeitliche Einschränkung zur Berechnung (Endedatum).

Beispiel:

```
SELECT *
FROM   q.Rules.AggregateIndicator( NULL, NULL, NULL )

SELECT *
FROM   q.Rules.AggregateIndicator( NULL, '01.01.2008', '30.06.2008' )

SELECT *
FROM   q.Rules.AggregateIndicator(
         'Category LIKE ''Nützlichkeit.Korrektheit''', NULL, NULL )
```

Routine	Rules.CheckSyntax
Typ	Gespeicherte Prozedur
Beschreibung	Überprüft die Syntax einer Regel und erzeugt bei fehlerhafter Syntax eine Ausnahme.

Parameter

@table	Tabellename, für die die Regel gelten soll.
@rule	Regel.
@action	Aktion (OnDelete, OnInsert, OnUpdate, OnChange, OnTimer).

Beispiel:

```
EXEC q.Rules.CheckSyntax
           'Sample.Persons.Customers',
           'IF Age = ''child'' THEN MaritalStatus = ''single''',
           NULL
```

Routine Rules.DeleteRule

Typ Gespeicherte Prozedur

Beschreibung Löscht eine Regel aus dem Repository.

Parameter

@code	Kurzbezeichnung der Regel.

Beispiel:

```
EXEC q.Rules.DeleteRule 'R9'
```

Routine	Rules.ExecRule
Typ	Gespeicherte Prozedur

Beschreibung Führt eine im Repository gespeicherte Regel explizit aus.

Parameter

@code	Kurzbezeichnung der Regel.

Beispiel:

```
EXEC q.Rules.ExecRule 'R9'
```

Routine	Rules.InsertRule
Typ	Gespeicherte Prozedur

Beschreibung Erstellt eine neue Regel und speichert diese im Repository.

Parameter

@code	Kurzbezeichnung der Regel.
@ruleEvent	Ereignis (OnDelete, OnInsert, OnUpdate, OnChange, OnTimer, OnApplication).
@ruleCondition	Bedingung.
@ruleAction	Aktion.
@category	Kategorie.
@status	Status (Active, Inactive).
@importance	Gewichtung der Regel (zwischen 0 und 100).
@table	Tabellenname, auf die sich die Regel bezieht.

Beispiel:

```
EXEC q.Rules.InsertRule
         'R9',
         'OnChange',
         'IF Age = ''child'' THEN MaritalStatus = ''single''',
         null,
         'Konsistenz',
         'active',
         90,
         'Sample.Persons.Customers'
```

Routine	Rules.Invalid
Typ	Gespeicherte Prozedur

Beschreibung Gibt alle Datensätze zurück, für die die Regel ungültig ist.

Parameter

@code	Kurzbezeichnung der Regel.
@sql	NULL oder SQL-Anweisung .

Beispiel:

```
EXEC q.Rules.Invalid 'R9',
                     NULL
EXEC q.Rules.Invalid 'R9',
                     'SELECT COUNT(*) FROM Sample.Persons.Customers'
EXEC q.Rules.Invalid 'R9',
                     'DELETE FROM Sample.Persons.Customers'
```

Routine Rules.Rules

Typ Tabellenwertfunktion

Beschreibung Gibt eine Liste mit allen Informationen zu allen im Repository gespeicherten Regeln aus.

Parameter

Keine

Beispiel:

```
SELECT *
FROM   q.Rules.Rules()
WHERE  Status = 'active' AND
       RuleEvent = 'OnChange'
```

Routine Rules.RulesTrend

Typ Tabellenwertfunktion

Beschreibung Gibt eine Liste aller im Repository gespeicherten Regeln aus, für die der Datenqualitätsindikator zu verschiedenen Zeitpunkten berechnet wurde.

Parameter

Keine

Beispiel:

```
SELECT *
FROM   q.Rules.RulesTrend()
WHERE  Date BETWEEN '01.01.2008' AND '30.06.2008' AND
       Indicator < 80.0
```

Routine	Rules.UpdateRule
Typ	Gespeicherte Prozedur

Beschreibung Ändert eine bereits bestehende Regel.

Parameter

@oldCode	Bisherige Kurzbezeichnung der Regel.
@newCode	Neue Kurzbezeichnung der Regel.
@ruleEvent	Ereignis (OnDelete, OnInsert, OnUpdate, OnChange, OnTimer, OnApplication).
@ruleCondition	Bedingung.
@ruleAction	Aktion.
@category	Kategorie.
@status	Status (Active, Inactive).
@importance	Gewichtung der Regel (zwischen 0 und 100).
@table	Tabellenname, auf die sich die Regel bezieht.

Beispiel:

```
EXEC q.Rules.UpdateRule
         'R9',
         'R9',
         'OnChange',
         'IF Age = ''child'' THEN MaritalStatus = ''single''',
         null,
         'Konsistenz',
         'inactive',
         90,
         'Sample.Persons.Customers'
```

Routine	Rules.UpdateStatistics
Typ	Gespeicherte Prozedur
Beschreibung	Ermittelt für jede Regel die Anzahl an Datensätzen, die der Regel nicht entsprechen bzw. ihr entsprechen und errechnet daraus einen Datenqualitätsindikator.
Parameter	
	keine

Beispiel:

```
EXEC q.Rules.UpdateStatistics
```

Routine Rules.Valid

Typ Gespeicherte Prozedur

Beschreibung Gibt alle Datensätze zurück, für die die Regel gültig ist.

Parameter

@code	Kurzbezeichnung der Regel.
@sql	NULL oder SQL-Anweisung .

Beispiel:

```
EXEC q.Rules.Valid 'R9',
                   NULL
EXEC q.Rules.Valid 'R9',
                   'SELECT COUNT(*) FROM Sample.Persons.Customers'
```

B.7 Control

Routine Control.AggregateIndicator

Typ Skalarwertfunktion

Beschreibung Liefert für mehrere Regeln einen aggregierten Gesamtindikator zur Beurteilung der Datenqualität zurück.

Parameter

@Where	Einschränkung der Regelmenge.

Beispiel:

```
SELECT q.Control.AggregateIndicator( NULL )
SELECT q.Control.AggregateIndicator( NULL )
SELECT q.Control.AggregateIndicator( 'Category LIKE ''Nützlichkeit%''' )
```

Routine Control.UpdateStatistics

Typ Gespeicherte Prozedur

Beschreibung Aktualisiert die Qualitätsindikatoren für jede Regel.

Parameter

keine

Beispiel:

```
EXEC q.Control.UpdateStatistics
```

B.8 Understand

Routine	Understand.ForeignKey
Typ	Tabellenwertfunktion
Beschreibung	Generische Funktion zum Analysieren von Primär-/Fremdschlüsselbeziehungen über Klassen, die die Schnittstelle *IForeignKey* implementieren.

Parameter

@table1	Erste Tabelle.
@columns1	Zu untersuchende Spalten der ersten Tabelle.
@where1	WHERE-Klausel für erste Tabelle zum Einschränken der zu untersuchenden Daten.
@table2	Zweite Tabelle.
@columns2	Zu untersuchende Spalten der zweiten Tabelle.
@where2	WHERE-Klausel für zweite Tabelle zum Einschränken der zu untersuchenden Daten.
@type	Vollständiger Klassenname.
@assemblyString	Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet.
@namedParameters	Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden.

Beispiel:

```
SELECT *
FROM   q.Understand.ForeignKey(
                'Sample.Persons.Customers', '*', NULL,
                'Sample.Sales.Orders',   '*', NULL,
                'FreD.Understand.Dependencies.ForeignKeyAnalyse',
                'Core',
                'Threshold=0.0' )
```

Routine	Understand.ForeignKeyAnalyse
Typ	Tabellenwertfunktion

Beschreibung	Funktion zum Analysieren von Primär-/Fremdschlüsselbeziehungen, indem instanzenbasiert die Schnittmengen an übereinstimmenden Daten zwischen den einzelnen Spalten jeder Tabelle ermittelt werden.

Parameter

@table1	Erste Tabelle.
@columns1	Zu untersuchende Spalten der ersten Tabelle.
@where1	WHERE-Klausel für erste Tabelle zum Einschränken der zu untersuchenden Daten.
@table2	Zweite Tabelle.
@columns2	Zu untersuchende Spalten der zweiten Tabelle.
@where2	WHERE-Klausel für zweite Tabelle zum Einschränken der zu untersuchenden Daten.

Beispiel:

```
SELECT *
FROM   q.Understand.ForeignKeyAnalyse(
                 'Sample.Persons.Customers', '*', NULL,
                 'Sample.Sales.Orders',   '*', NULL )
WHERE RightValues = 0 OR LeftValues = 0
```

Routine	Understand.ForeignKeyTextMatchingAnalyse
Typ	Tabellenwertfunktion

Beschreibung	Funktion zum Analysieren von Primär-/Fremdschlüsselbeziehungen, indem schemabasiert die Spaltennamen über einen Textvergleichsalgorithmus auf Übereinstimmung überprüft werden.

Parameter

@table1	Erste Tabelle.
@columns1	Zu untersuchende Spalten der ersten Tabelle.
@where1	WHERE-Klausel für erste Tabelle zum Einschränken der zu untersuchenden Daten.
@table2	Zweite Tabelle.
@columns2	Zu untersuchende Spalten der zweiten Tabelle.
@where2	WHERE-Klausel für zweite Tabelle zum Einschränken der zu untersuchenden Daten.

Beispiel:

```
SELECT *
FROM   q.Understand.ForeignKeyTextMatchingAnalyse(
                'Sample.Persons.Customers', '*', NULL,
                'Sample.Sales.Orders',   '*', NULL )
```

Routine	Understand.PrimaryKey
Typ	Tabellenwertfunktion

Beschreibung Generische Funktion zum Erkennen von möglichen Primärschlüsseln über Klassen, die die Schnittstelle *IPrimaryKey* implementieren.

Parameter

@table	Tabellenname.
@columns	Zu untersuchende Spalten der Tabelle.
@where	WHERE-Klausel zum Einschränken der zu untersuchenden Daten.
@type	Vollständiger Klassenname.
@assemblyString	Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet.
@namedParameters	Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden.

Beispiel:

```
SELECT *
FROM   q.Understand.PrimaryKey(
                'Sample.Sales.Articles', '*', NULL,
                'FreD.Understand.Dependencies.PrimaryKeyAnalyse',
                'Core', 'Threshold=0.4' )
```

Routine	Understand.PrimaryKeyAnalyse
Typ	Tabellenwertfunktion

Beschreibung Funktion zum Erkennen von möglichen Primärschlüsseln über die *PrimaryKeyAnalyse*-Klasse.

Parameter

@table	Tabellenname.
@columns	Zu untersuchende Spalten der Tabelle.
@where	WHERE-Klausel zum Einschränken der zu untersuchenden Daten.

Beispiel:

```
SELECT *
FROM   q.Understand.PrimaryKeyAnalyse(
            'Sample.Sales.Articles',
            '*',
            NULL )
```

Routine Understand.PropertiesBenfordsLaw

Typ Aggregatfunktion

Beschreibung Funktion zum Ermitteln der Häufigkeiten der einzelnen Ziffern einer Zahl zur Überprüfung von Benford's Law.

Parameter

@value	Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesBenfordsLaw( ListPrice )
FROM   Sales.Articles
```

Routine Understand.PropertiesBlankCount

Typ Aggregatfunktion

Beschreibung Funktion zum Ermitteln der Anzahl leerer Einträge (nicht NULL).

Parameter

@value	Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesBlankCount( Title )
FROM   Persons.Customers
```

Routine	Understand.PropertiesCountNotNull
Typ	Aggregatfunktion
Beschreibung	Funktion zum Ermitteln der Anzahl Einträge, die nicht NULL enthalten.

Parameter

@value	Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesCountNotNull( Title )
FROM   Persons.Customers
```

Routine Understand.PropertiesCountNull

Typ Aggregatfunktion

Beschreibung Funktion zum Ermitteln der Anzahl Einträge, die NULL enthalten.

Parameter

@value	Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesCountNull( Title )
FROM   Persons.Customers
```

Routine	Understand.PropertiesDataPropery
Typ	Aggregatfunktion

Beschreibung Generische Aggregatfunktion zum Aufrufen von Klassen, die die Schnittstelle *IDataProperty* implementieren.

Parameter

@value	Konkatenierte Zeichenkette aus Name der Klasse, die *IDataProperty* implementiert, Assemblyname, benannte Parameter und zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesDataProperty(
            '{FreD.Understand.Properties.Number.MeanProperty|
             Core|
             TrimmedCount=10}' +
             COALESCE( CAST( ListPrice AS NVARCHAR(50) ), '{NULL}' ) )
FROM   Sales.Articles
```

Routine	Understand.PropertiesDataTypeRecognised
Typ	Aggregatfunktion
Beschreibung	Funktion zum Ermitteln des Datentyps der Spalte anhand des Dateninhaltes.

Parameter

@value	Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesDataTypeRecognised( Weight )
FROM   Sales.Articles
```

Routine Understand.PropertiesGeometricMean

Typ Aggregatfunktion

Beschreibung Errechnet den geometrischen Mittelwert.

Parameter

@value	Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesGeometricMean( ListPrice )
FROM   Sales.Articles
WHERE  ListPrice < 10
```

Routine Understand.PropertiesHarmonicMean

Typ Aggregatfunktion

Beschreibung Errechnet den harmonischen Mittelwert.

Parameter

@value	Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesHarmonicMean( ListPrice )
FROM   Sales.Articles
```

Routine Understand.PropertiesIntegerCount

Typ Aggregatfunktion

Beschreibung Ermittelt die Anzahl an Ganzzahlen.

Parameter

@value	Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesIntegerCount( ListPrice )
FROM   Sales.Articles
```

Routine Understand.PropertiesMax

Typ Aggregatfunktion

Beschreibung Ermittelt die fünf maximalen Werte einer Spalte.

Parameter

@value	Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesMax( ListPrice )
FROM   Sales.Articles
```

Routine Understand.PropertiesMaxDecimalPlaces

Typ Aggregatfunktion

Beschreibung Ermittelt die maximale Anzahl an Nachkommastellen.

Parameter

@value	Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesMaxDecimalPlaces( ListPrice )
FROM   Sales.Articles
```

Routine Understand.PropertiesMaxTextCount

Typ Aggregatfunktion

Beschreibung Ermittelt die Anzahl an Zeichenketten, die am längsten sind.

Parameter

@value	Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesMaxTextCount( Name )
FROM   Sales.Articles
```

Routine Understand.PropertiesMaxTextLength

Typ Aggregatfunktion

Beschreibung Ermittelt die Länge der längsten Zeichenkette.

Parameter

@value	Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesMaxTextLength( Name )
FROM   Sales.Articles
```

Routine Understand.PropertiesMean

Typ Aggregatfunktion

Beschreibung Berechnet den arithmetischen Mittelwert.

Parameter

@value	Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesMean( ListPrice )
FROM   Sales.Articles
```

Routine Understand.PropertiesMin

Typ Aggregatfunktion

Beschreibung Ermittelt die fünf minimalen Werte.

Parameter

@value	Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesMin( ListPrice )
FROM   Sales.Articles
```

Routine Understand.PropertiesMinDecimalPlaces

Typ Aggregatfunktion

Beschreibung Ermittelt die minimale Anzahl an Nachkommastellen.

Parameter

@value	Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesMaxDecimalPlaces( ListPrice )
FROM   Sales.Articles
```

Routine Understand.PropertiesMinTextCount

Typ Aggregatfunktion

Beschreibung Ermittelt die Anzahl an Zeichenketten, die am kürzesten sind.

Parameter

@value	Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesMinTextCount( Name )
FROM   Sales.Articles
```

Routine Understand.PropertiesMinTextLength

Typ Aggregatfunktion

Beschreibung Ermittelt die Länge der kürzesten Zeichenkette.

Parameter

@value	Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesMinTextLength( Name )
FROM   Sales.Articles
```

Routine	Understand.PropertiesPrimaryKeyCandidate
Typ	Aggregatfunktion
Beschreibung	Ermittelt anhand der Werte für eine Spalte, ob diese als Primärschlüssel geeignet ist.

Parameter

@value	Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesSquareSum( ListPrice )
FROM   Sales.Articles
```

Routine Understand.PropertiesProduct

Typ Aggregatfunktion

Beschreibung Errechnet das Produkt der Werte der numerischen Spalte.

Parameter

@value	Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesProduct( ListPrice )
FROM   Sales.Articles
WHERE  ListPrice < 10
```

Routine Understand.PropertiesRange

Typ Aggregatfunktion

Beschreibung Errechnet die Spannbreite für eine numerische Spalte.

Parameter

@value	Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesRange( ListPrice )
FROM   Sales.Articles
```

Routine Understand.PropertiesSquareSum

Typ Aggregatfunktion

Beschreibung Errechnet die Quadratsumme für eine numerische Spalte.

Parameter

@value	Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesSquareSum( ListPrice )
FROM   Sales.Articles
```

Routine Understand.PropertiesStdDev

Typ Aggregatfunktion

Beschreibung Errechnet die Standardabweichung für eine numerische Spalte.

Parameter

@value	Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesStdDev( ListPrice )
FROM   Sales.Articles
```

Routine Understand.PropertiesStdErr

Typ Aggregatfunktion

Beschreibung Errechnet den Standardfehler für eine numerische Spalte.

Parameter

@value	Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesStdErr( ListPrice )
FROM   Sales.Articles
```

Routine Understand.PropertiesSum

Typ Aggregatfunktion

Beschreibung Errechnet die Summe für eine numerische Spalte.

Parameter

@value	Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesSum( ListPrice )
FROM   Sales.Articles
```

Routine Understand.PropertiesVariance

Typ Aggregatfunktion

Beschreibung Errechnet die Varianz für eine numerische Spalte.

Parameter

@value	Zu analysierende Spalte.

Beispiel:

```
SELECT q.Understand.PropertiesVariance( ListPrice )
FROM   Sales.Articles
```

Routine	Understand.PropertyListData
Typ	Tabellenwertfunktion

Beschreibung Generische Funktion zum Ermitteln aller Daten-Eigenschaften für Spalten einer Tabelle aus einer Klasse einer Assembly, die *IDataPropertyList* implementiert.

Parameter

@table	Tabellenname.
@columns	Zu untersuchende Spalten der Tabelle.
@where	WHERE-Klausel zum Einschränken der zu untersuchenden Daten.
@type	Vollständiger Klassenname.
@assemblyString	Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet.
@namedParameters	Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden.

Beispiel:

```
SELECT "Column", Name, Value, Description
FROM q.Understand.PropertyListData(
               'Sample.Sales.Articles',
               '*',
               NULL,
               'FreD.Understand.Properties.DataPropertyListGeneric',
               'Core',
               NULL )
ORDER BY "Column", Name
```

Routine	Understand.PropertyListDataGeneric
Typ	Tabellenwertfunktion

Beschreibung Ermittelt für Spalten einer Tabelle alle Daten-Eigenschaften aus Klassen einer Assembly, die *IDataProperty* implementieren.

Parameter

@table	Tabellenname.
@columns	Zu untersuchende Spalten der Tabelle.
@where	WHERE-Klausel zum Einschränken der zu untersuchenden Daten.
@type	Vollständiger Klassenname.
@assemblyString	Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet.
@namedParameters	Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden.

Beispiel:

```
SELECT "Column", Name, Value, Description
FROM q.Understand.PropertyListDataGeneric(
                         'Sample.Sales.Articles',
                         *',
                         null, 'core' )
ORDER BY "Column", Name
```

Routine	Understand.PropertyListSchema
Typ	Tabellenwertfunktion

Beschreibung Generische Funktion zum Ermitteln aller Schema-Eigenschaften für Spalten einer Tabelle aus einer Klasse einer Assembly, die *ISchemaPropertyList* implementiert.

Parameter

@table	Tabellenname.
@columns	Zu untersuchende Spalten der Tabelle.
@type	Vollständiger Klassenname.
@assemblyString	Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet.
@namedParameters	Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden.

Beispiel:

```
SELECT "Column", Name, Value, Description
FROM q.Understand.PropertyListSchema(
                'Sample.Sales.Articles',
                '*',
                'FreD.Understand.Properties.SchemaPropertyListGeneric',
                'Core',
                NULL )
ORDER BY "Column", Name
```

Routine	Understand.PropertyListSchemaGeneric
Typ	Tabellenwertfunktion

Beschreibung Ermittelt für Spalten einer Tabelle alle Schema-Eigenschaften über ADO.NET.

Parameter

@table	Tabellenname.
@columns	Zu untersuchende Spalten der Tabelle.

Beispiel:

```
SELECT "Column", Name, Value, Description
FROM q.Understand.PropertyListSchemaGeneric(
                  'Sample.Sales.Articles',
                  '*' )
ORDER BY "Column", Name
```

Routine	Understand.RuleInduction
Typ	Tabellenwertfunktion

Beschreibung Generische Funktion zum Suchen nach Regeln im Datenbestand.

Parameter

@table	Tabellenname.
@columns	Zu untersuchende Spalten der Tabelle.
@where	WHERE-Klausel zum Einschränken der zu untersuchenden Daten.
@type	Vollständiger Klassenname.
@assemblyString	Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet.
@namedParameters	Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden.

Beispiel:

```
SELECT "Rule", Confidence, Support, Frequency
FROM q.Understand.RuleInduction(
     'Sample.Persons.Customers',
     'Title, Age, MaritalStatus, Sex',
     null,
     'FreD.Understand.Rules.AprioriRuleInduction',
     'WekaRuleInduction',
     'MinItemFrequency=3, Delta=0.05' )
```

Routine	Understand.RuleInductionOperator
Typ	Tabellenwertfunktion
Beschreibung	Funktion zum Suchen nach Größer-, Kleiner-, Gleich-Regeln für numerische Spalten im Datenbestand.

Parameter

@table	Tabellenname.
@columns	Zu untersuchende Spalten der Tabelle.
@where	WHERE-Klausel zum Einschränken der zu untersuchenden Daten.

Beispiel:

```
SELECT "Rule", Support, Frequency
FROM q.Understand.RuleInductionOperator(
     'Sample.Sales.Articles',
     '*',
     NULL )
```

B.9 Improve

Routine Improve.Deduplicate

Typ Gespeicherte Prozedur

Beschreibung Führt die Duplikaterkennung für eine Tabelle durch, indem ein im Repository abgelegter Partitioner und ein Detector geladen werden. Über den Partitioner werden jeweils immer zwei Datensätze aus einer Tabelle gelesen und über den Detector auf Gleichheit getestet. Sind sie identisch, so wird in der Spalte „q.Group“ die gleiche Kennung eingetragen. Nach der eigentlichen Duplikaterkennung sucht die Prozedur noch alle transitiven Duplikate und kennzeichnet diese entsprechend.

Parameter

@partitionerName	Name des im Repository gespeicherten Partitioners.
@detectorName	Name des im Repository gespeicherten Detectors.
@comparators	Spezifische *IMatching*-Klassen für jede zu überprüfende Spalte getrennt durch das ‚|‘-Zeichen oder NULL um die im Detector deklarierte generische IMatching-Klasse zu verwenden (*JaroApproximateMatching*).

Beispiel:

```
EXEC q.Improve.Deduplicate
  'PartitionerPersonsCustomers',
  'DetectorPersonsCustomers',
  'FreD.Text.Matching.JaroWinklerLynchApproximateMatching|
   FreD.Text.Matching.TypewriterApproximateMatching|
   FreD.Number.Matching.SimpleNumberMatching, MaxDistance=10|
   FreD.Distance.Matching.GeoDistanceMatching,Country=US,MaxDistance=50|
   null|
   FreD.Text.Matching.ExactTextMatching,StartIndex=0,Length=2'
```

Routine	Improve.DeduplicateIncremental
Typ	Gespeicherte Prozedur

Beschreibung

Führt die Duplikaterkennung für zwei Tabellen durch, indem ein im Repository abgelegter Partitioner und ein Detector geladen werden. Beide Tabellen müssen die gleiche Struktur besitzen. Über den Partitioner wird jeweils ein Datensatz aus beiden Tabellen gelesen und über den Detector auf Gleichheit getestet. Sind sie identisch, so wird in der Spalte „q.Group“ die gleiche Kennung eingetragen. Nach der eigentlichen Duplikaterkennung sucht die Prozedur noch alle transitiven Duplikate und kennzeichnet diese entsprechend.
Die Prozedur ist zum automatischen Erkennen von Duplikaten beim Ändern/Einfügen eines Datensatzes durch eine Regel gedacht.

Parameter

@tableIncremental	Name der Tabelle, in der die abzugleichenden Datensätze gespeichert sind (Struktur muss identisch sein mit Tabelle des Partitioners).
@partitionerName	Name des im Repository gespeicherten Partitioners.
@detectorName	Name des im Repository gespeicherten Detectors.
@comparators	Spezifische *IMatching*-Klassen für jede zu überprüfende Spalte getrennt durch das ‚|‘-Zeichen oder NULL um die im Detector deklarierte generische IMatching-Klasse zu verwenden (*JaroApproximateMatching*).

Beispiel:

```
EXEC q.Rules.InsertRule
     'RD20',
     'OnInsert',
     null,
     'EXEC q.Improve.DeduplicateIncremental
            ''q.temp.inserted'',
            ''PartitionerPersonsCustomers'',
            ''DetectorPersonsCustomers'',
           null;
```

```
      IF EXISTS(
                  SELECT [q.Group]
                  FROM q.temp.inserted
                  WHERE [q.Group] IS NOT NULL
                )
          RAISERROR( ''Duplicate in customer table'', 18, 1 );',
     'Konsistenz',
     'active',
     0,
     'Sample.Persons.Customers'


INSERT INTO Persons.Customers
           ( CustomerId, Name, Street, PostalCode, City, State )
     VALUES
           ( 201, 'Mary Connors', 'Havel Street', '8125', 'Mack', 'CO' )


INSERT INTO Persons.Customers
           ( CustomerId, Name, Street, PostalCode, City, State )
     VALUES
           ( 202, 'John Smith', 'Water Street', '8125', 'Mack', 'CO' )
```

Routine	Improve.DeduplicationBuildDetector
Typ	Gespeicherte Prozedur

Beschreibung Erzeugt einen Detector, der jeweils zwei Datensätze auf Gleichheit überprüft und von den beiden Prozeduren *Deduplicate* und *DeduplicateIncremental* verwendet wird.

Parameter

@detectorName	Name des Detectors, unter dem dieser im Repository gespeichert werden soll.
@type	Vollständiger Klassenname des Detectors.
@assemblyString	Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet.
@namedParameters	Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden.

Beispiel:

```
EXEC q.Improve.DeduplicationBuildDetector
        'DetectorPersonsCustomers',
        'FreD.Improve.Deduplication.FellegiSunterDetector',
        'Core',
        'Threshold=0.75,
         {GenericComparator=FreD.Text.Matching.JaroApproximateMatching},
         Trace=false'
```

Routine	Improve.DeduplicationBuildPartitioner
Typ	Gespeicherte Prozedur

Beschreibung Erzeugt einen Partitioner, der eine Tabelle in Partitionen aufteilt, indem er der Tabelle die drei Spalten „q.ID“ (eindeutige Satzkennung), „q.MatchKey“ (Partitionskennung) und „q.Group“ (Duplikatkennung) hinzufügt. Die Prozedur wird von den beiden Prozeduren *Deduplicate* und *DeduplicateIncremental* verwendet.

Parameter

@table	Tabellenname.
@matchKey	Anweisung zur Erzeugung des Matchkey.
@fields	Spalten, die verglichen werden sollen.
@where	WHERE-Klausel zum Einschränken der zu untersuchenden Daten.
@partitionerName	Name des Partitioners, unter dem dieser im Repository gespeichert werden soll.
@type	Vollständiger Klassenname des Partitioners.
@assemblyString	Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet.
@namedParameters	Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden.

Beispiel:

```
EXEC q.Improve.DeduplicationBuildPartitioner
         'Sample.Persons.Customers',
         'q.Text.PatternsCharSort( SUBSTRING( Name, 1, 5 ) )',
         'Name, Street, StreetNumber, PostalCode, City, State',
         NULL,
         'PartitionerPersonsCustomers',
         'FreD.Improve.Deduplication.WindowingPartitioner',
         'Core',
         NULL
```

Routine	Improve.DeduplicationConcat
Typ	Aggregatfunktion
Beschreibung	Konkateniert alle Werte einer Gruppe zu einer gemeinsamen Zeichenkette mit dem ‚|'-Zeichen als Trenner.

Parameter

@value	Spaltenname.

Beispiel:

```
SELECT "q.Group",
       q.Improve.DeduplicationConcat( Name )
FROM Persons.Customers
WHERE "q.Group" IS NOT NULL
GROUP BY "q.Group"
```

Routine Improve.DeduplicationDeleteDetector

Typ Gespeicherte Prozedur

Beschreibung Löscht einen Detector aus dem Repository.

Parameter

@detectorName	Name des Detectors.

Beispiel:

```
EXEC q.Improve.DeduplicationDeleteDetector 'DetectorPersonsCustomers'
```

Routine Improve.DeduplicationDeletePartitioner

Typ Gespeicherte Prozedur

Beschreibung Löscht einen Partitioner aus dem Repository.

Parameter

@partitionerName	Name des Partitioners.

Beispiel:

```
EXEC q.Improve.DeduplicationDeletePartitioner
             'PartitionerPersonsCustomers'
```

Routine	Improve.DeduplicationDiscard
Typ	Aggregatfunktion
Beschreibung	Liefert nur dann kein NULL zurück, wenn alle Werte der Gruppe identisch sind.

Parameter

@value	Spaltenname.

Beispiel:

```
SELECT "q.Group",
       q.Improve.DeduplicationDiscard( Title ),
       q.Improve.DeduplicationConcat( Name )
FROM   Persons.Customers
WHERE  "q.Group" IS NOT NULL
GROUP BY "q.Group"
```

Routine Improve.DeduplicationFirst

Typ Aggregatfunktion

Beschreibung Liefert den ersten Wert innerhalb der Gruppe zurück.

Parameter

@value	Spaltenname.

Beispiel:

```
SELECT "q.Group",
       q.Improve.DeduplicationFirst( Name )
FROM   Persons.Customers
WHERE  "q.Group" IS NOT NULL
GROUP BY "q.Group"
```

Routine	Improve.DeduplicationFuse
Typ	Aggregatfunktion

Beschreibung Generische Aggregatfunktion zum Verwenden einer Klasse, die die Schnittstelle *IFuse* implementiert.

Parameter

@value	Spaltenname.

Beispiel:

```
SELECT "q.Group",
       q.Improve.DeduplicationFuse(
            '{FreD.Improve.Deduplication.Fuse.Concat|Core}'+
            COALESCE( Name, '{NULL}') )
FROM   Persons.Customers
WHERE  "q.Group" IS NOT NULL
GROUP BY "q.Group"
```

Routine	Improve.DeduplicationInfoDetector
Typ	Gespeicherte Prozedur
Beschreibung	Gibt Informationen zu einem im Repository gespeicherten Detector aus.

Parameter

@detectorName	Name des Detectors.

Beispiel:

```
SELECT q.Improve.DeduplicationInfoDetector( 'DetectorPersonsCustomers' )
```

Routine	Improve.DeduplicationInfoPartitioner
Typ	Gespeicherte Prozedur
Beschreibung	Gibt Informationen zu einem im Repository gespeicherten Partitioner aus.

Parameter

@partitionerName	Name des Partitioners.

Beispiel:

```
SELECT q.Improve.DeduplicationInfoPartitioner(
                        'PartitionerPersonsCustomers' )
```

Routine Improve.DeduplicationLast

Typ Aggregatfunktion

Beschreibung Liefert den letzten Wert innerhalb der Gruppe zurück.

Parameter

@value	Spaltenname.

Beispiel:

```
SELECT "q.Group",
       q.Improve.DeduplicationLast( Name )
FROM   Persons.Customers
WHERE  "q.Group" IS NOT NULL
GROUP BY "q.Group"
```

Routine Improve.DeduplicationLongestText

Typ Aggregatfunktion

Beschreibung Liefert die längste Zeichenkette innerhalb der Gruppe zurück.

Parameter

@value	Spaltenname.

Beispiel:

```
SELECT "q.Group",
       q.Improve.DeduplicationLongestText( Name )
FROM   Persons.Customers
WHERE  "q.Group" IS NOT NULL
GROUP BY "q.Group"
```

Routine Improve.DeduplicationMax

Typ Aggregatfunktion

Beschreibung Liefert den größten Wert innerhalb der Gruppe zurück.

Parameter

@value	Spaltenname.

Beispiel:

```
SELECT "q.Group",
       q.Improve.DeduplicationMax( PostalCode )
FROM   Persons.Customers
WHERE  "q.Group" IS NOT NULL
GROUP BY "q.Group"
```

Routine Improve.DeduplicationMaxFrequency

Typ Aggregatfunktion

Beschreibung Liefert den am häufigsten vorkommenden Wert innerhalb der Gruppe zurück.

Parameter

@value	Spaltenname.

Beispiel:

```
SELECT "q.Group",
       q.Improve.DeduplicationMaxFrequency( Street )
FROM   Persons.Customers
WHERE  "q.Group" IS NOT NULL
GROUP BY "q.Group"
```

Routine Improve.DeduplicationMean

Typ Aggregatfunktion

Beschreibung Liefert den arithmetischen Mittelwert einer Gruppe von Zahlen zurück.

Parameter

@value	Spaltenname.

Beispiel:

```
SELECT "q.Group",
       q.Improve.DeduplicationMean( ListPrice )
FROM   Sales.Articles
WHERE  "q.Group" IS NOT NULL
GROUP BY "q.Group"
```

Routine Improve.DeduplicationMin

Typ Aggregatfunktion

Beschreibung Liefert den kleinsten Wert innerhalb der Gruppe zurück.

Parameter

@value	Spaltenname.

Beispiel:

```
SELECT "q.Group",
       q.Improve.DeduplicationMin( PostalCode )
FROM   Persons.Customers
WHERE  "q.Group" IS NOT NULL
GROUP BY "q.Group"
```

Routine Improve.DeduplicationShortestText

Typ Aggregatfunktion

Beschreibung Liefert die kürzeste Zeichenkette innerhalb der Gruppe zurück.

Parameter

@value	Spaltenname.

Beispiel:

```
SELECT "q.Group",
       q.Improve.DeduplicationShortestText( Name )
FROM   Persons.Customers
WHERE  "q.Group" IS NOT NULL
GROUP BY "q.Group"
```

Routine Improve.DeduplicationStdDev

Typ Aggregatfunktion

Beschreibung Liefert die Standardabweichung einer Gruppe von Zahlen zurück.

Parameter

@value	Spaltenname.

Beispiel:

```
SELECT "q.Group",
       q.Improve.DeduplicationStdDev( ListPrice )
FROM   Sales.Articles
WHERE  "q.Group" IS NOT NULL
GROUP BY "q.Group"
```

Routine Improve.DeduplicationSum

Typ Aggregatfunktion

Beschreibung Liefert die Summe einer Gruppe von Zahlen zurück.

Parameter

@value	Spaltenname.

Beispiel:

```
SELECT "q.Group",
       q.Improve.DeduplicationSum( ListPrice )
FROM   Sales.Articles
WHERE  "q.Group" IS NOT NULL
GROUP BY "q.Group"
```

Routine Improve.DeduplicationVariance

Typ Aggregatfunktion

Beschreibung Liefert die Varianz einer Gruppe von Zahlen zurück.

Parameter

@value	Spaltenname.

Beispiel:

```
SELECT "q.Group",
       q.Improve.DeduplicationVariance( ListPrice )
FROM   Sales.Articles
WHERE  "q.Group" IS NOT NULL
GROUP BY "q.Group"
```

Routine	Improve.PredictorBuild
Typ	Gespeicherte Prozedur

Beschreibung Erzeugt einen Predictor zur Vorhersage eines Spaltenwertes über Eingabespalten.

Parameter

@table	Tabellenname.
@fields	Eingabespalten.
@fieldClass	Vorhersagespalte.
@where	WHERE-Klausel zum Einschränken der zu untersuchenden Daten.
@predictionName	Name des Predictors, unter dem dieser im Repository gespeichert werden soll.
@type	Vollständiger Klassenname des Partitioners.
@assemblyString	Vollständige Beschreibung der Identität einer Assembly. Wird nur der Name der Assembly angegeben, so werden Standardwerte für Version (0.0.0.0), Culture (neutral) und PublicKeyToken (NULL) verwendet.
@namedParameters	Benannte Parameter, die zum Setzen von Eigenschaften der Klasseninstanz verwendet werden.

Beispiel:

```
EXEC q.Improve.PredictorBuild
            'Sample.Persons.Customers',
            'Age, MaritalStatus, Sex, State',
            'Rating',
            null,
            'PredictorCustomerRating',
            'FreD.Improve.Prediction.Id3Predictor',
            'WekaPrediction',
            null
```

Routine	Improve.PredictorBuildLookup
Typ	Gespeicherte Prozedur

Beschreibung Erzeugt einen LookupPredictor zur Vorhersage eines Spaltenwertes über Eingabespalten, indem aus einer Nachschlagetabelle der Zielwert ausgelesen wird.

Parameter

@table	Tabellenname.
@fields	Eingabespalten.
@fieldClass	Vorhersagespalte.
@where	WHERE-Klausel zum Einschränken der zu untersuchenden Daten.
@predictionName	Name des Predictors, unter dem dieser im Repository gespeichert werden soll.

Beispiel:

```
EXEC q.Improve.PredictorBuildLookup
                 'q.Lookup."City.US"',
                 'PostalCode',
                 'City',
                 null,
                 'PredictorCity'
```

Routine	Improve.PredictorBuildZeroR
Typ	Gespeicherte Prozedur

Beschreibung Erzeugt einen ZeroRPredictor zur Vorhersage eines Spaltenwertes über Eingabespalten.

Parameter

@table	Tabellenname.
@fields	Eingabespalten.
@fieldClass	Vorhersagespalte.
@where	WHERE-Klausel zum Einschränken der zu untersuchenden Daten.
@predictionName	Name des Predictors, unter dem dieser im Repository gespeichert werden soll.

Beispiel:

```
EXEC q.Improve.PredictorBuildZeroR
            'Sample.Persons.Customers',
            'Age, MaritalStatus, Rating',
            'Sex',
            null,
            'PredictorCustomerSex'
```

Routine	Improve.PredictorDelete
Typ	Gespeicherte Prozedur
Beschreibung	Löscht einen im Repository gespeicherten Predictor.

Parameter

@predictionName	Name des Predictors, unter dem dieser im Repository gespeichert ist.

Beispiel:

```
EXEC q.Improve.PredictorDelete  'PredictorCustomerSex'
```

Routine	Improve.PredictorInfo
Typ	Gespeicherte Prozedur
Beschreibung	Gibt Informationen zu einem im Repository gespeicherten Predictor aus.

Parameter

@predictionName	Name des Predictors, unter dem dieser im Repository gespeichert ist.

Beispiel:

```
PRINT q.Improve.PredictorInfo( 'PredictorCity')
```

Routine Improve.PredictorPredict

Typ Skalarwertfunktion

Beschreibung Sagt auf Grundlage eines vorher erstellten Predictors den Wert für eine Zielspalte vorher.

Parameter

@predictionName	Name des Predictors, unter dem dieser im Repository gespeichert werden soll.
@values	Eingabewerte zur Vorhersage.

Beispiel:

```
SELECT Name, Rating,
       q.Improve.PredictorPredict(
                                  'PredictorCustomerRating',
                                  q.Tools.Param( Age, 0 ) +
                                  q.Tools.Param( MaritalStatus, 0 ) +
                                  q.Tools.Param( Sex, 0 ) +
                                  q.Tools.Param( State, 1 ) )
FROM  Persons.Customers
WHERE Rating <> q.Improve.PredictorPredict(
                                  'PredictorCustomerRating',
                                  q.Tools.Param( Age, 0 ) +
                                  q.Tools.Param( MaritalStatus, 0 ) +
                                  q.Tools.Param( Sex, 0 ) +
                                  q.Tools.Param( State, 1 ) )



SELECT PostalCode, City,
       q.Improve.PredictorPredict(
               'PredictorCity', q.Tools.Param( PostalCode, 1 ) )
FROM  Persons.Customers
WHERE City <> q.Improve.PredictorPredict(
               'PredictorCity', q.Tools.Param( PostalCode, 1 ) )
```

B.10 Tools

Routine	Tools.AddPlugin
Typ	Gespeicherte Prozedur
Beschreibung	Registriert ein Plugin zur Verwendung im Datenqualitäts-Framework.

Parameter

@script	Name des Predictors, unter dem dieser im Repository gespeichert werden soll.
@assemblyName	Name, unter der das Skript geladen werden soll.
@references	Zusätzliche Assemblies, die zum Übersetzen des Skripts benötigt werden. Bei mehrerem Assemblies werden diese durch das ‚|'-Zeichen getrennt.

Beispiel:

```
EXEC q.Tools.AddPlugin
         'C:\FreD\samples\Predictor.cs',
         'SamplePredictor',
         'C:\FreD\Core.dll|System.Data.dll'

EXEC q.Tools.AddPlugin
       'namespace MyEncoder
       {
           public class UpperTextPreparer :
                        FreD.Text.Encoder.GenericTextPreparer
           {
               protected override string DoEncode( string text )
               {
                   base.IgnoreCase = true;
                   return base.DoEncode( text );
               }
            }
       }',
       'SampleEncoder',
       'C:\FreD\Core.dll'
```

Routine	Tools.BuildFrequencyTable
Typ	Gespeicherte Prozedur

Beschreibung Erstellt automatisch eine Häufigkeitstabelle für alle in den Spalten vorkommenden Werte.

Parameter

@frequencyTable	Name der Häufigkeitstabelle, die im Repository mit dem Präfix „frq“ angelegt wird.
@table	Tabellenname.
@fields	Spalten, für dessen Werte die Häufigkeitstabelle erstellt wird.
@where	WHERE-Klausel zum Einschränken der zu untersuchenden Daten.
@distinct	Wenn true, wird ein Token innerhalb eines Datensatzes nur einmal gezählt.
@useTokenizer	Wenn true, Verwendung der *GenericTokenizer*-Klasse, ansonsten kein Auftrennen.

Beispiel:

```
EXEC q.Tools.BuildFrequencyTable
        'PersonsCustomers',
        'Sample.Persons.Customers',
        'Title, Name, Street, StreetNumber, PostalCode, City, State',
        null, 1, 0
```

Routine	Tools.Log
Typ	Gespeicherte Prozedur
Beschreibung	Prozedur zum Protokollieren ins Eventlog, in die Datenbank oder in eine Datei.

Parameter

@method	Bestimmt ob in das Eventlog, in die Datenbank oder in eine CSV-Datei protokolliert wird (Eventlog: „event:“, Datenbank: „database:“, Datei: „file:“).
@category	Kategorie des Protokolleintrags (z.B. Info, Warning, Error).
@info	Zu protokollierender Text.

Beispiel:

```
EXEC q.Tools.Log  'file://c:\Sample.log', 'Sample', 'Duplicate found!'
EXEC q.Tools.Log  'event:', 'Sample', 'Duplicate found!'
EXEC q.Tools.Log  'database://SampleLog', 'Sample', 'Duplicate found!'
```

Routine	Tools.LogInvalid
Typ	Gespeicherte Prozedur
Beschreibung	Prozedur zum Protokollieren von Datensätzen ins Eventlog, in die Datenbank oder in eine Datei, die eine bestimmte Regel verletzen.

Parameter

@method	Bestimmt ob in das Eventlog, in die Datenbank oder in eine CSV-Datei protokolliert wird (Eventlog: „event:“, Datenbank: „database:“, Datei: „file:“).
@category	Kategorie des Protokolleintrags (z.B. Info, Warning, Error).
@info	Zu protokollierender Text.
@code	Kurzbezeichnung der Regel.

Beispiel:

```
EXEC q.Tools.LogInvalid 'file://c:\Sample.log',
                        'Sample',
                        'Rule R9 violated', 'R9'
```

Routine	Tools.Param
Typ	Skalarwertfunktion
Beschreibung	Funktion zum Übergeben mehrerer Werte über einen einzigen Übergabeparameter für die Funktion *Improve.PredictorPredict*.

Parameter

@value	Zusammenzusetzende Werte.
@lastParam	True für den letzten Übergabewert.

Beispiel:

```
SELECT q.Tools.Param( Title, 0 ) +
       q.Tools.Param( Name, 0 ) +
       q.Tools.Param( City, 1 )
FROM   Persons.Customers
```

Routine	Tools.RelativeFrequency
Typ	Skalarwertfunktion

Beschreibung Berechnet die relative Häufigkeit bestimmter Werte einer Spalte. Wird u.a. verwendet, um Regeln zu formulieren, die sich auf mehrere Datensätze einer Tabelle beziehen.

Parameter

@table	Tabellenname.
@field	Spaltenname.
@values	Werte, für die die relative Häufigkeit berechnet werden soll.

Beispiel:

```
SELECT q.Tools.RelativeFrequency(
         'Sample.Persons.Customers',
         'Age',
         '''senior'', ''junior''' )


EXEC q.Rules.InsertRule
         'RELATION_GENDER',
         'OnInsert',
         'BOOL{q.Tools.RelativeFrequency(
              ''Sample.Persons.Customers'',
              ''Sex'', ''''''female'''''' )} <
          CAST( 0.70 AS REAL )',
         'EXEC q.Tools.Log
              ''file://c:\Sample.log'',
              ''Statistic'',
              ''Relation of female customers shrinking below 70%''',
         'Konsistenz',
         'active',
         5,
         'Sample.Persons.Customers'

INSERT INTO Persons.Customers
           ( CustomerId, Name, PostalCode, City, State, Sex )
VALUES
           ( 209, 'Mary Connors', 'Mack', 'CO', 'male' )
```

Routine	Tools.SendMail
Typ	Gespeicherte Prozedur

Beschreibung Prozedur zum Versenden einer E-Mail.

Parameter

@host	SMTP-Servername.
@from	E-Mail-Absendeadresse.
@to	E-Mail-Empfängeradresse.
@subject	Betreff.
@body	Inhalt der E-Mail.

Beispiel:

```
EXEC q.Tools.SendMail
          'mail.sample.de',
          'database@sample.de',
          'marketing@sample.de',
          'DQ (Sample.Persons.Customers)',
          'Less than 70% of female customers'


EXEC q.Rules.InsertRule
         'RELATION_GENDER',
         'OnInsert',
           'BOOL{q.Tools.RelativeFrequency(
              ''Sample.Persons.Customers'',
              ''Sex'', ''''''female'''''' )} <
          CAST( 0.70 AS REAL )',
           'q.Tools.SendMail
              ''mail.sample.de'',
              ''database@sample.de'',
              ''marketing@sample.de'',
              ''DQ (Sample.Persons.Customers)'',
              ''Less than 70% of female customers'';
             PRINT ''Mail sent!''',
         'Konsistenz',
         'active',
         5,
         'Sample.Persons.Customers'

INSERT INTO Persons.Customers
           ( CustomerId, Name, PostalCode, City, State, Sex )
VALUES
           ( 209, 'Mary Connors', 'Mack', 'CO', 'male' )
```

Routine	Tools.SendMailInvalid
Typ	Gespeicherte Prozedur

Beschreibung Versendet die Datensätze, die eine Regel verletzen, per E-Mail.

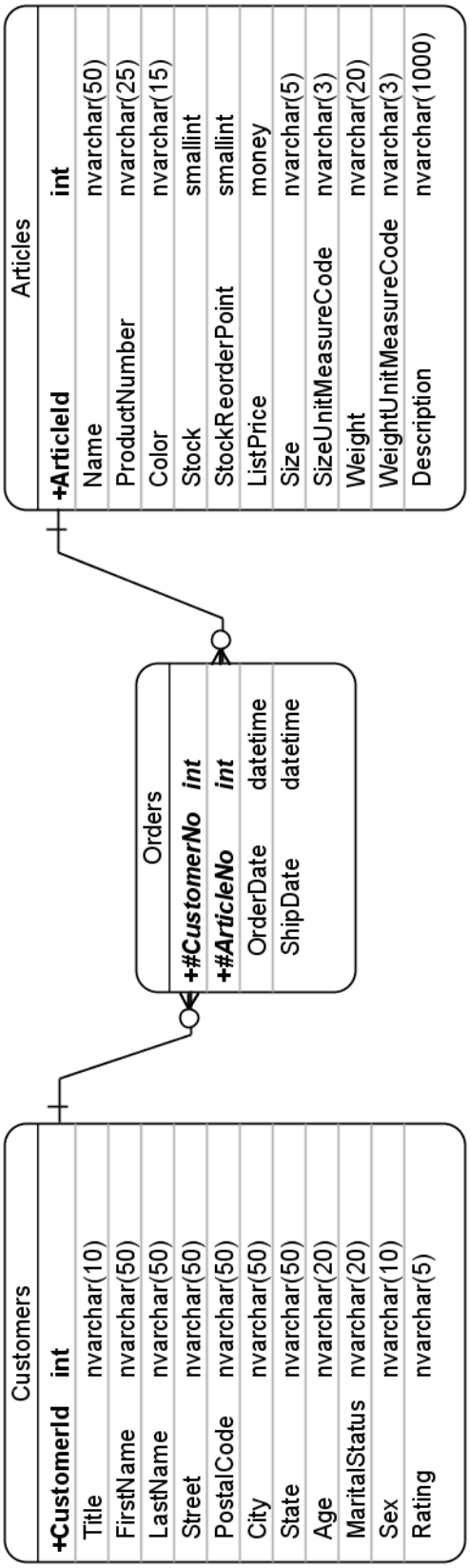
Parameter

@host	SMTP-Servername.
@from	E-Mail-Absendeadresse.
@to	E-Mail-Empfängeradresse.
@subject	Betreff.
@body	Inhalt der E-Mail.
@code	Kurzbezeichnung der Regel.

Beispiel:

```
EXEC q.Tools.SendMailInvalid
          'mail.sample.de',
          'database@sample.de',
          'dataSteward@sample.de',
          'DQ (Sample.Persons.Customers)',
          'MaritalStatus=single, if Age=child',
          'R9'


EXEC q.Rules.InsertRule
         'R9',
         'OnChange',
         'IF Age = ''child'' THEN MaritalStatus = ''single''',
         null,
         'Konsistenz',
         'active',
         90,
         'Sample.Persons.Customers'
```

C Datenmodell für Beispieldatenbank



Literatur

Armstrong, M.: An Overview of the Issues Related to the Use of Personal Identifiers, Catalogue no. 85-602-XIE; Statistics Canada; Ottawa

Astrahan, M.M. u.a.: System R: Relational Approach to Database Management; erschienen in: ACM Transactions on Database Systems, Vol. 1, No. 2; ACM Press; New York; 1976; S. 97-137

Augustin, S.: Der Stellenwert des Wissensmanagement im Unternehmen; erschienen in: Mandl, H.; Reinmann-Rothmeier, G. (Hrsg.): Wissensmanagement: Informationszuwachs - Wissensschwund? Die strategische Bedeutung des Wissensmanagements; München; Oldenbourg; 2000; S. 159-168

Backhaus, K. u.a.: Multivariate Analysemethoden; 10. Auflage; Springer Verlag; Berlin u.a.; 2003

Bange, C.: Das neue Gesicht der Datenintegration; Business Application Reseach Center (BARC); http://www.competence-site.de/datenbanken.nsf/9D457579637252C7C1256E6F0033EFDC/$File/datenintegration_barc_0304.pdf; 2004

Bauer, A.; Günzel H. (Hrsg.): Data Warehouse Systeme, 1. Auflage; dpunkt Verlag; Heidelberg; 2001

Baxter, R.; Christen, P.; Churches, T.: A Comparison of Fast Blocking Methods for Record Linkage; erschienen in: Proceedings of the Workshop on Data Cleaning, Record Linkage and Object Consolidation at the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; Washington DC; 2003; o. S.

Baeza-Yates, R.; Ribeiro-Neto, B.: Modern Information Retrieval; Addison-Wesley; Harlow; 1999

Balzert, H.: Lehrbuch der Software-Technik - Software-Entwicklung, 2. Auflage; Spektrum Akademischer Verlag; Heidelberg, Berlin; 2000

Batini, C.; Scannapieco, M.: Data Quality - Concepts, Methodologies and Techniques; Springer-Verlag; Heidelberg; 2006

Bauckmann, J.: Efficiently Identifying Inclusion Dependencies in RDBMS; erschienen in: 18. Workshop über Grundlagen von Datenbanken (GI-Workshop); Wittenberg; 2006

Bauckmann, J.; Leser, U.; Naumann, F.: Efficiently Computing Inclusion Dependencies for Schema Discovery; erschienen in Workshop InterDB (with ICDE06); Atlanta; 2006

Behme, W.; Nietzschmann, S.: Strategien zur Verbesserung der Datenqualität im DWH-Umfeld; erschienen in: HMD - Praxis der Wirtschaftsinformatik 02/2006, Heft 247; dpunkt Verlag; Heidelberg; 2006; S. 43-53

Beesley, K. R.: Language identifier: A computer program for automatic natural-language identification on on-line text; erschienen in: Proceedings of the 29th Annual Conference of the American Translators Association; 1988; S. 47-54

Bilenko, M.: Learnable Similarity Functions and Their Applications to Record Linkage and Clustering – Doctoral Dissertation Proposal, University of Texas, Austin, 2003

Bilke, A.: Duplicate-based Schema Matching, Dissertation; Technische Universität Berlin, Fakultät Elektrotechnik und Informatik; Berlin; 2007

Bilke, A.; Bleiholder, J.; Böhm, C.; Draba, K.; Naumann, F.; Weis, M.: Automatic Data Fusion with HumMer; erschienen in: Proceedings of the 31st Very Large Data Bases (VLDB) Conference; Trondheim, Norwegen; 2005

Bleiholder, J.;Naumann, F.: Declarative Data Fusion - Syntax, Semantics and Implementation; erschienen in: Eder, J.; Haav, H.-M.; Kalja, A.; Penjam, J. (Hrsg.): Advances in Databases and Information Systems 9th East European Conference, Tallinn, Estonia; September 12-15, 2005; Proceedings; Lecture Notes in Computer Science (LNCS), Vol. 3631, S. 58-73

Bleiholder, J.; Naumann, F.:FUSE BY: Syntax und Semantik zur Informationsfusion in SQL. Informatik 2004 Workshop über Dynamische Informationsfusion.; Ulm, Germany; September 2004

Bleiholder, J.: A Relational Operator Approach to Data Fusion, 31st International Conference on Very Large Data Bases (VLDB) 2005 PhD Workshop; Trondheim; Norway; September 2005

Borgelt, C.; Kruse, R.: Induction of Association Rules: Apriori Implementation; erschienen in: 15th Conference on Computational Statistics (Compstat 2002); Berlin; S. 395-400

Borgelt, C.: Efficient Implementations of Apriori and Eclat; erschienen in: Workshop of Frequent Item Set Mining Implementations (FIMI 2003); Melbourne, Florida; 2003

Borgelt, C.: An Implementation of the FP-growth Algorithm; erschienen in: Workshop Open Source Data Mining Software (OSDM'05); Chicago; 2005; S. 1-5

Borgelt, C.: Keeping Things Simple: Finding Frequent Item Sets by Recursive Elimination; erschienen in: Workshop Open Source Data Mining Software (OSDM'05); Chicago; 2005; S. 66-70

Borthwick, A.: A Maximum Entropy Approach to Named Entity Recognition; Ph.D. Thesis; New York University; New York; 1999

Borthwick, A.: Probabilistic Record Linkage Model derived from Training Data; United States Patent Application Publication; o.A.; 2003

Bortz, J.: Statistik für Human- und Sozialwissenschafter; 6. Auflage; Springer Medizin Verlag; Heidelberg; 2005

Califf, M. E.: Relational Learning Techniques for Natural Language Information Extraction, Dissertation; University of Texas; Austin; 1998, S. 5

Caspers, P.; Gebauer, M.: Reproduzierbare Messung von Datenqualität mit Hilfe des DQ-Messtools WestLB-DIME; erschienen in: Dadam, P; Reichert, M. (Hrsg.): Informatik 2004 - Informatik verbindet, Band 1, Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V.; Bonner Köllen Verlag, Bonn, 2005; S. 234-238

Caumanns, J.: A Fast and Simple Stemming Algorithm; Technical Report Nr. TR-B-99-16 des Fachbereichs Informatik der Freien Universität Berlin; 1999

Chamberlin, D.D.; Boyce, R.F.: SEQUEL: Structured English Query Language; erschienen in: Proceedings of the 1974 ACM SIGFIDET workshop on Data description, access and control: Data models: Data-structure-set versus relational; Ann Arbor, Michigan; 1974; S. 249-263

Charras, C.; Lecroq, T.: Handbook of Exact String Matching Algorithms, King's College Publications, o.A., 2004

Chieu, H.; Ng, H.: Named Entity Recognition with a Maximum Entropy Approach; erschienen in: Proceedings of the Seventh Conference on Natural Language Learning; Edmonton, Canada; 2003

Chinchor, N.: MUC-7 Named Entity Task Definition; Version 3.5; http://www.itl.nist.gov/iaui/894.02/related_projects/muc/proceedings/ne_task.html; 1997

Chisholm, M.: How to build a Business Rules Engine; Morgan Kaufmann; San Francisco; 2004

Christen, P.: Improving data linkage and deduplication quality through nearest-neighbour based blocking; erschienen in: Proceedings of thirteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07); San Jose, California; 2007; o. S.

Christen, P.; Churches, T.: Febrl - Freely extensible biomedical record linkage (Manual Release 0.3); http://sourceforge.net/project/downloading.php?groupname=febrl&filename=febrldoc-0.3.pdf; 2005; S. 66

Cilibrasi, R.; Vitanyi, P.M.B.: Clustering by Compression; erschienen in: IEEE Transactions on Information Theory, Vol. 51, Issue 4; o.O.; 2005; S. 1523-1545

Cochinwala, M.; Dalal, S.; Elmagarmid, A. K.; Verykios, V. S.: Record matching: Past, present and future; Technical Report TR-01-013; Purdue University, Department of Computer Sciences; 2001

Codd, E. F.: A Relational Model of Data for Large Shared Data Banks; erschienen in: Communications of the ACM, Vol. 13, Issue 6; ACM Press, New York, USA; 1970; S. 377-387

Connolly, T.M.; Begg, C.E.: Database Systems, Fourth Edition; Addison-Wesley Pearson Education; Harlow, England; 2005

Cordts, S.: Datenbankkonzepte in der Praxis; Addison-Wesley; München; 2002

Cordts, S.; Müller, B.: Dublettenbereinigung nach dem Record Linkage Algorithmus; erschienen in: Cremers, A. u.a. (Hrsg.): Beiträge der 35. Jahrestagung der Gesellschaft für Informatik e.V., Informatik 2005 - Informatik LIVE!, Band 2, Bonner Köllen Verlag, Bonn, 2005; S. 428-432

Damm, H. M.: Total anti-symmetrische Quasigruppen; Phillips-Universiät, Fachbereich Mathematik und Informatik; Marburg/Lahn; 2004

Dasu, T.; Johnson, T.: Exploratory Data Mining and Data Cleaning; John Wiley & Sons; Hoboken, New Jersey; 2003

Date, C.J.: An Introduction to Database Systems, Sixth Edition; Addison-Wesley; Reading, Massachusetts; 1995

Dippold, R.; Meier, A.; Schnider, W.; Schwinn, K.: Unternehmensweites Datenmanagement, 4. Auflage; Vieweg; Braunschweig, Wiesbaden; 2005

Dittrich, K.R.; Gatziu, S.: Aktive Datenbanksysteme - Konzepte und Mechanismen, 2. Auflage; dpunkt Verlag; Heidelberg; 2000

Dittrich, K.R.; Geppert, A.: Component Database Systems: Introduction, Foundations, and Overview; erschienen in: Dittrich, K.R.; Geppert, A. (Eds.): Component Database Systems; Morgan Kaufmann; San Francisco; 2001; S. 1-28

Do, H.; Rahm, E.: COMA A system for flexible combination of Schema Matching Approaches; erschienen in: Proceedings of the 28.th Very Large Data Bases (VLDB) Conference; Hong Kong; 2002

Dörre, J.; Gerstl, P.; Seiffert, R.: Volltextsuche und Text Mining; erschienen in: Carstensen, K.-U. u.a. (Hrsg.): Computerlinguistik und Sprachtechnologie, 2. Auflage; Spektrum Akademischer Verlag; Heidelberg; 2004

Dougherty, J.; Kohavi, R.; Sahami, M.: Supervised and Unsupervised Discretization of Continuous Features; erschienen in: Machine Learning: Proceedings of the 12th International Conference; Morgan Kaufmann; San Francisco; 1995; S. 194-202

Dunn, H.L.: Record Linkage; erschienen in: American Journal of Public Health, Vol. 36; New York; 1946; S. 1412–1416.

Elfeky, M. G.; Elmagarmid, A. K.; Ghanem, T. M.: Towards Quality Assurance of Government Databases: A Record Linkage Web Service; Demo at DG.O'2002, The 3rd National Conference on Digital Government Research; Los Angeles, California; 2002

Elfeky, M. G.; Verykios, V.; Elmagarmid, A. K.: TAILOR: A Record Linkage Toolbox; erschienen in: Proceedings of the International Conference on Data Engineering (ICDE); San Jose; 2002

Elmagarmid, A.K.; Ipeirotis, P.G.; Verykios, V.S.: Duplicate Record Detection: A Survey; erschienen in: IEEE Transactions on Knowledge and Data Engineering, Vol. 19, No. 1; IEEE Computer Society; Washington, DC; 2007; S. 1-16

Elmasri, R.; Navathe, S. B.: Grundlagen von Datenbanksystemen; 3. Auflage, Pearson Education; München; 2005

Elmasri, R.; Navathe, S. B.: Fundamentals of Database Systems; Third Edition, Addison-Wesley; Reading, Massachusetts; 2000

English, L.P.: Improving Data Warehouse and Business Information Quality; Wiley Computer Publishing; New York; 1999

Ferber, R.: Information Retrieval; dpunkt Verlag; Heidelberg; 2003

Fortier, P.: SQL 3 - Implementing the SQL Foundation Standard; McGraw-Hill; New York u.a.; 1999

Fox, C.: Lexical Analysis and Stoplists; erschienen in: Frakes, W.B.; Baeza-Yates, R. (Hrsg.): Information Retrieval - Data Structures & Algorithms; Prentice-Hall; Upper Saddle River, New Jersey; 1992

Frakes, W.B.: Introduction to Information Storage and Retrieval Systems; erschienen in: Frakes, W.B.; Baeza-Yates, R. (Hrsg.): Information Retrieval - Data Structures & Algorithms; Prentice-Hall; Upper Saddle River, New Jersey; 1992

Frakes, W.B.: Stemming Algorithms; erschienen in: Frakes, W.B.; Baeza-Yates, R. (Hrsg.): Information Retrieval - Data Structures & Algorithms; Prentice-Hall; Upper Saddle River, New Jersey; 1992

Friedl, J.E.F.: Reguläre Ausdrücke, 2. Auflage; O'Reilly; 2003; Köln

Friedrich, D.: Einfach soll es sein - bei hoher Datenqualität; erschienen in: is report; 11. Jahrgang, 4/2007; Oxygon Verlag; München; S. 28-29

Fritschi, H.: A Component Framework to Construct Active Database Management Systems, Dissertation; Wirtschaftswissenschaftliche Fakultät der Universität Zürich; Zürich; 2002

Galhardas, H.; Florescu, D.; Shasa, D.; Simon, E.: An Extensible Framework for Data Cleaning (extended version); INRIA Technical Report; Sophia Antipolis; 1999

Galhardas, H.; Florescu, D.; Shasa, D.; Simon, E.; Saita, C.-A.: Declarative Data Cleaning: Language, Model and Algorithms; erschienen in: Proceeding of the 27th VLDB Conference; Roma, Italy; 2001

Gallian, J.A.: Error Detection Methods; erschienen in: ACM Computing Surveys; September 1996, Vol. 28, No. 3; ACM Press; New York, USA; 1996; S. 504-517

Garcia-Molina, H.; Ullman, J.D.; Widom, J.: Database Systems: The Complete Book; Pearson Education International; Upper Saddle River, New Jersey; 2002

Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: Entwurfsmuster; Addison-Wesley; München; 1996

Gill, L.: The Oxford Medical Record Linkage System, Record Linkage Techniques; erschienen in: Proceedings of an International Workshop and Exposition; Arlington; 1997

Gill, L.: Methods for Automatic Record Matching and Linkage and their Use in National Statistics, National Statistics Methodological, Series No. 25; National Statistics; London; 2001

Goiser, K.; Christen, P.: Towards Automated Record Linkage; erschienen in: Christen, P.; Kennedy, P. J.; Jiuyong, L.; Simoff, S. J.; Williams, G. J. (Hrsg.): Data Mining and Analytics 2006, Proceedings of the Fifth Australasian Data Mining Conference, Conferences in Research and Practice in Information Technology (CRPIT), Vol. 61.; ACS Press; Sydney; S. 23-31

Gravano, L.; Ipeirotis, P.G.; Jagadish, H.V.; Koudas, N.; Muthukrishnan, S.; Pietarinen, L.; Srivastava, D.: Using q-grams in a DBMS for Approximate String Processing; erschienen in: IEEE Data Engineering Bulletin, Vol. 24, No. 4; IEEE Computer Society; Washington, DC; 2001; S. 28-34

Gravano, L.; Ipeirotis, P.G.; Jagadish, H.V.; Koudas, N.; Muthukrishnan, S.; Srivastava, D.: Approximate String Joins in a Database (Almost) for Free; erschienen in: Proceedings of the 27th Very Large DataBase (VLDB) Conference; Rom, Italien; 2001

Gravano, L.; Ipeirotis, P.G.; Koudas, N.; Srivastava, D.: Text Joins for Data Cleansing and Integration in an RDBMS; erschienen in: Proceedings of the 19th IEEE International Conference on Data Engineering (ICDE 2003); Bangalore, Indien; 2003

Gray, J.: The Revolution in Database Architecture, Keynote talk; erschienen in: ACM SIGMOD 2004; Paris; 2004

Gray, J.; Compton, M.: A Call To Arms; erschienen in: ACM Queue; Vol. 3, No. 3; ACM; New York, USA; 2005; S. 30-38

Greenfield, J.; Short, K.: Software Factories - Moderne Software-Architekturen mit SOA, MDA, Patterns und agilen Methoden; Redline; Heidelberg; 2006

Grimmer, U.; Hinrichs, H.: Datenqualitätsmanagement mit Data-Mining-Unterstützung; erschienen in: Hildebrand, K. (Hrsg.): Business Intelligence, HMD - Praxis der Wirtschaftsinformatik 12/2001, Heft 222; dpunkt Verlag; Heidelberg; 2001; S. 70-80

Grishman, R.; Sundheim, B.: Message Understanding Conference - 6: A Brief History; erschienen in: Proceedings of the 16th conference on Computational linguistics; Copenhagen, Denmark; 1996

Gu, L.; Baxter, R.A.; Vickers, D.; Rainsford, C.: Record Linkage: Current Practice and Future Directions, Technical Report 03/83; CSIRO Mathematical and Information Sciences; Canberra, Australia; 2003

Gulutzan, P.; Pelzer, T.: SQL-99 Complete, Really; R&D Books; Lawrence, Kansas; 1999

Härder, T.: DBMS Architecture - the Layer Model and its Evolution; erschienen in: Datenbank-Spektrum, 5. Jahrgang, Heft 13; dpunkt Verlag; Heidelberg; 2005; S. 45-57

Härder, T.: Rahm, E.: Datenbanksysteme - Konzepte und Techniken der Implementierung; Springer Verlag; Berlin, Heidelberg; 1999

Hafer, M. A.; Weis, S. F.: Word Segmentation by Letter Successor Variety; erschienen in: Information Storage and Retrieval, Vol. 10, Issue 11-12; Pergamon Press; Oxford; 1974; S. 371-385

Halama, A.: Flache Satzverarbeitung; erschienen in: Carstensen, K.-U. u.a. (Hrsg.): Computerlinguistik und Sprachtechnologie, 2. Auflage; Spektrum Akademischer Verlag; Heidelberg; 2004

von Halle, B.; Goldberg, L.: The Business Rule Revolution; Happy About; Silicon Valley; 2006

Han, J.; Kamber, M.: Data Mining - Concepts and Techniques; Morgan Kaufmann; San Francisco; 2001

Hausotter, A.: Datenorganisation; erschienen in: Disterer, G.; Fels, F.; Hausotter, A.: Taschenbuch der Wirtschaftsinformatik, 2. Auflage; Carl Hanser Verlag; München, Wien; 2003; S. 209-237

Helfert, M.: Proaktives Datenqualiätsmanagement in Data-Warehouse-Systemen, Dissertation; logos-Verlag; Berlin; 2002

Herbst, H.; Knolmayer, G.: Ansätze zur Klassifikation von Geschäftsregeln, Arbeitsbericht Nr. 46; Institut für Wirtschaftsinformatik der Universität Bern; Bern; 1994

Hernandez, M. A.; Stolfo, S. J.: Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem; Data Mining and Knowledge Discovery, Vol. 2, No. 1; Kluwer Academic; Boston; 1998; S. 9-37

Heller, M.: Approximative Indexierungstechniken für historische deutsche Textvarianten; erschienen in: Historical Social Research / Historische Sozialforschung, Vol. 31 (2006), No. 3; Zentrum für Historische Sozialforschung e.V.; Köln; 2006

Hettich, S.; Hippner, H.: Assoziationsanalyse; erschienen in: Hippner, H. u.a. (Hrsg.): Handbuch Data Mining im Marketing, 1. Auflage,;Vieweg; Braunschweig; 2001; S. 427-463

Hildebrandt, K.: Datenqualität im Supply Chain Management; erschienen in: Dadam, P; Reichert, M. (Hrsg.): Informatik 2004 - Informatik verbindet, Band 1, Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V.; Bonner Köllen Verlag, Bonn, 2005; S. 239-243

Hill, T.P.: The first-digit phenomenon; erschienen in: American Scientist, Vol. 86, No. 4; Research Triangle Park, North Carolina; 1998; S. 358-363

Hipp, J.: Datenqualität - Ein zumeist unterschätzer Erfolgsfaktor; erschienen in: Dadam, P; Reichert, M. (Hrsg.): Informatik 2004 - Informatik verbindet, Band 1, Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V.; Bonner Köllen Verlag, Bonn, 2005; S. 232-233

Hirschberg, D. S.: A linear space algorithm for computing maximal common subsequences; erschienen in: Communications of the ACM; Vol. 18, Issues 6; ACM Press, New York; 1975

Huhtala, Y; Kärkkäinen, J.; Porkka, P.; Toivonen, H.: Efficient Discovery of Functional and Approximate Dependencies using Partitions; erschienen in: 14th International Conference on Data Engineering (ICDE'98); IEEE Computer Society Press; Orlando, Florida; 1998; S. 392-401

Huhtala, Y; Kärkkäinen, J.; Porkka, P.; Toivonen, H.: TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies; erschienen in: The Computer Journal Vol. 42 No. 2; Oxford, United Kingdom; 1999, S. 100-111

Jenkins, M.-C.; Smith, D.: Conservative stemming for search and indexing; erschienen in: Proceedings of the 28.th ACM SIGIR Conference; Salvador, Brasilien; 2005

Kemper, A.; Eickler, A.: Datenbanksysteme, 6. Auflage; Oldenbourg Wissenschaftsverlag; München; 2006

Kirtland, J.: Identification Numbers and Check Digit Schemes; The Mathematical Association of America; 2001

Klein, D.; Smarr, J.; Nguyen, H.; Manning, C.: Named Entity Recognition with Character-Level Models; erschienen in: Proceedings of the Seventh Conference on Natural Language Learning; Edmonton, Canada; 2003

Kowalski, G.J.; Maybury, M.T.: Information Storage and Retrieval Systemes - Theory and Implementation, Second Edition; Kluwer Academic; Norwell, Massachusetts; 2000

Krovetz, R.: Viewing Morphology as an Inference Process; erschienen in: Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval; 1993; S. 191-203

Kübel, J.; Grimmer, U.; Hipp, J.: Regelbasierte Ausreißersuche zur Datenqualitätsanalyse; erschienen in: Datenbank-Spektrum, 5. Jahrgang, Heft 14; dpunkt Verlag; Heidelberg; 2005; S. 22-29

Lait, A.J., Randell, B.: An Assessment of Name Matching Algorithms, Technical Report; Department of Computing Science, University of Newcastle upon Tyne; Newcastle; 1993

Lee, Y.W.; Pipino, L.L.; Funk, J.D.; Wang, R.Y.: Journey to Data Quality; MIT Press; Cambridge, Massachusetts; 2006

Lee, Y.W.; Strong, D.M.: Knowing-Why About Data Processes and Data Quality; erschienen in: Journal of Management Information Systems; Vol. 20, No. 3; 2004; M.E. Sharpe; Armonk, New York, USA; S. 13-39

Leser, U.; Naumann, F.: Informationsintegration; dpunkt Verlag; Heidelberg; 2007

Li, M.; Li, X.; Ma, B.; Vitanyi, P.M.B.: The Similarity Metric; erschienen in: Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms; Baltimore; 2003; S. 863-872

Lockemann, P.C.; Dittrich, K.R.: Architektur von Datenbanksystemen; dpunkt Verlag; Heidelberg; 2004

Low, W.L.; Lee, M.L.; Ling, T.W.: A Knowledge-Based Approach for Duplicate Elimination in Data Cleaning; erschienen in: Information Systems, Vol. 26, Issue 8 (December 2001); Elsevier Science; Oxford, UK; S. 585-606

Lüssem, J.; Tasev, P.; Tiedemann-Muhlack, M.: Ein lernendes System zur Verbesserung der Datenqualität und Datenqualitätsmessung; erschienen in: Cremers, A. u.a. (Hrsg.): Beiträge der 35. Jahrestagung der Gesellschaft für Informatik e.V., Informatik 2005 - Informatik LIVE!, Band 2, Bonner Köllen Verlag, Bonn, 2005; S. 418-422

Mathes, T.; Bange, C.; Keller, P.: Software im Vergleich: Datenqualitätsmanagement, Studie des Business Application Research Center (BARC); Oxygon Verlag; München; 2005

Melnik, S.; Garcia-Molina, H.; Rahm, E.: Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching; erschienen in: Proceedings of the 18th ICDE; San Jose; 2002

Melton, J.; Simon, A.R.: SQL:1999 - Understanding Relational Language Components; Morgan Kaufmann; San Francisco; 2002

Merkl, R.; Waack, S.: Bioinformatik interaktiv; Wiley-VCH; Weinheim; 2003

Michael, J.: Doppelgänger gesucht - Ein Programm für kontextsensitive phonetische Stringumwandlung; erschienen in: c't Magazin für Computer und Technik 25/1999. S. 252ff

Michael, J.: Mit Sicherheit – Prüfziffernverfahren auf Modulo-Basis; erschienen in: c't magazin für computertechnik; Ausgabe 7/97, S. 264ff

Michael, J.: Nicht wörtlich genommen - Schreibweisentolerante Suchroutinen in dBase implementiert; erschienen in: c't Magazin für Computer und Technik 10/1988. S. 126ff

Michel, C.: Cardinal , nominal or ordinal similarity measures in comparative evaluation of information retrieval process; erschienen in: Actes du congrès LREC 2000 Second international conference on langage ressource and evaluation; Athènes, 2000; S. 1509-1513

Mikheev, A.; Moens, M.; Grover, C.: Named Entity Recognition without Gazetteers; erschienen in: Proceedings of the Ninth Conference of the European Chapter of the Association for Computational Linguistics (EACL); Bergen, Norway; 1999; S. 1-8

Müller, H.; Freytag, J.-C.: Problems, Methods, and Challenges in Comprehensive Data Cleansing, Technical Report HUB-IB-164; Humboldt Universität; Berlin; 2003; S. 6f

Müller, H.; Weis, M.; Bleiholder, J.; Leser, U.: Erkennen und Bereinigen von Datenfehlern in naturwissenschaftlichen Daten; erschienen in: Datenbank-Spektrum, 5. Jahrgang, Heft 15; dpunkt Verlag; Heidelberg; 2005; S. 26-35

Monge, A. E.; Elkan, C. P.: The field matching problem: Algorithms and applications; erschienen in: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining; Portland, Oregon; 1996; S. 267-270

Naumann, F.: Datenqualität; erschienen in: Informatik-Spektrum; Vol. 30, No. 1, Februar 2007; Springer Verlag; Heidelberg, Berlin; 2007, S. 27-31

Naumann, F.: Informationsintegration - Antrittsvorlesung, Heft 134; Humboldt-Universität zu Berlin; Berlin; 2004

Naumann, F.; Häussler, M.: Declarative Data Merging with Conflict Resolution; erschienen in: Proceedings of the 7th International Conference on Information Quality (ICIQ-02); Cambridge, Massachusetts, USA; 2002

Naumann, F.; Leser, U.; Freytag, J. C.: Quality-driven Integration of Heterogeneous Information Systems, Informatik-Bericht Nr. 117; Humboldt-Universität zu Berlin; Berlin; 1999

Naumann, F.; Rolker, C.: Assessment Methods for Information Quality Criteria, Informatik-Bericht Nr. 138; Humboldt-Universität zu Berlin; Berlin; 2000

Naumann, F.; Roth, M.: Information Quality: How good are Off-The-Shelf DBMS?; erschienen in: Proceedings of the 9th International Conference on Information Quality (ICIQ-04); Cambridge, Massachusetts, USA; S. 260-274

Navarro, G.: A Guided Tour to Approximate String Matching; erschienen in: ACM Computing Surveys Vol. 33 Issue 1; ACM Press; New York; 2001, S. 31-88

Navarro, G.; Raffinot, M.: Flexible Pattern Matching in Strings; Cambridge University Press; Cambridge; 2002

Newcombe, H. B.; Kennedy, J. M.: Record linkage: making maximum use of the discriminating power of identifying information; Communications of the ACM archive Volume 5 , Issue 11; New York; 1962; S. 563-566

Object Management Group Common Facilities RFP-4: Common business objects and business object facility; OMG TC Document CF/96-01-04; 1996; Framingham

Olson, J.: Data Quality - Accuracy Dimension; Morgan Kaufman Publishers; San Francisco; 2003

Pfeifer, U., Poersch, T., Fuhr, N.: Retrieval Effectiveness of Proper Name Search Methods; erschienen in: Information Processing and Management, Vol. 32, No. 6, S. 667-679

Pfeifer, U. u.a.: Searching Proper Names in Databases; University of Dortmund; Dortmund; 1994

Porter, E. H.; Winkler, W. E.: Approximate String Comparison and its Effect on an Advanced Record Linkage System; U.S. Bureau of the Census, Research Report; Washington; 1997

Porter, M. F.: An algorithm for suffix stripping; erschienen in: Sparck Jones, K.; Willett, P (Hrsg.).: Readings in Information Retrieval; Morgan Kaufmann Publishers; San Francisco; 1997; S. 313-316

Pyle, D.: Data Preparation for Data Mining; Morgan Kaufmann; San Diego; 1999

Rahm, E.; Do, H.H.: Data Cleaning: Problems and Current Approaches; IEEE Bulletin of the Technical Committee on Data Engineering, Vol. 23, No. 4, December 2000; Washington, DC; S. 3-13

Ratcliff, J.W.; Metzener, D.E.: Pattern Matching: The Gestalt Approach; erschienen in: Dr. Dobb's Journal, No. 141; Boulder, Colorado; 1988; S. 46-51

Redman, T.: Data Quality for the Information Age; Artech House; Norwood; 1996

Reussner, R.; Hasselbring, W.: Handbuch der Software-Architektur; dpunkt Verlag; 2006; Heidelberg

Robertson, S.: Understanding Inverse Document Frequency: On theoretical arguments for IDF; erschienen in: Journal of Documentation, Vol. 60, No. 5; Emerald Group; Bradford, United Kingdom; 2004; S. 503-520

Saake, G.; Heuer, A.; Sattler, K.-U.: Datenbanken: Implementierungstechniken, 2. Auflage; mitp-Verlag; Bonn; 2005

Scannapieco, M.; Missier, P.; Batini, C.: Data Quality at a Glance; erschienen in: Datenbank-Spektrum, 5. Jahrgang, Heft 14; dpunkt Verlag; Heidelberg; 2005; S. 6-14

Scheibl, H.-J.: Grundlagen Phonetischer Datenbankabgleich; Fachhochschule für Technik und Wirtschaft Berlin; http://www2.f1.fhtw-berlin.de/scheibl/phonet/index.htm?./grundl/grundl02.htm; abgerufen am 13.08.2007

Schlesinger, M.: ALFRED - Konzepte und Prototyp einer aktiven Schicht zur Automatisierung von Geschäftsregeln, Dissertation; Rechts- und Wirtschaftswissenschaftliche Fakultät der Universität Bern; Bern; 2000

Schneider, M.: Implementierungskonzepte für Datenbanksysteme; Springer Verlag; Berlin, Heidelberg; 2004;

Schöning, U.: Algorithmik, Spektrum Akademischer Verlag, Heidelberg, 2001

Schulz, R.-H.: Codierungstheorie; 2. Auflage; Vieweg Verlag; Wiesebaden; 2003

Sedgewick, R.: Algorithmen, 2. Auflage; Addison-Wesley; München; 2002

Sparck-Jones, K.: A statistical interpretation of term specificity and its application in retrieval; erschienen in: Journal of Documentation, Vol. 28, No. 1; Emerald Group; Bradford, United Kingdom; 1972; S. 11-21

Sparck-Jones, K.; Willett, P. (Hrsg.): Readings in Information Retrieval; Morgan Kaufmann Publishers; San Francisco; 1997

Srikant, R.; Agrawal, R.: Mining Quantitative Association Rules in Large Relational Tables; erschienen in: Proceedings of the ACM SIGMOD Conference on Management of Data; Montreal; 1996; S. 1-12

Stallkamp, M.: Ziele und Entscheiden - Die Goalgetter-Methode und Goalgetter-Software, Diss.; Universität Duisburg-Essen, Wirtschaftswissenschaften; Essen; 2006

Tabeling, P.: Softwaresysteme und ihre Modellierung; Springer-Verlag; Berlin, Heidelberg; 2006

Toral, A.: DRAMNERI: a free knowledge based tool to Named Entity Recognition; erschienen in: Proceedings of the 1st Free Software Technologies Conference; A Coruña, Spanien; 2005; S. 27-32

Türker, C.: SQL:1999 & SQL:2003 - Objektrelationales SQL, SQLJ & SQL/XML; dpunkt Verlag; 2003

Türker, C.; Gertz, M.: Semantic integrity support in SQL:1999 and commercial (object-) relational database management systems; erschienen in: The VLDB Journal - The international Journal on Very Large Data Bases, Vol. 10, Issue 4; Springer Verlag; New York; 2001; S. 241-269

Verykios, V.; Elmagarmid, A.: Automating the Approximate Record Matching Process; Purdue University; West Lafayette; 1999

Vossen, G.: Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme, 4. Auflage; Oldenbourg Wissenschaftsverlag; München; 2000

Wagner, N. R.; Putter, P. S.: Error Detecting Decimal Digits; erschienen in: Communications of the ACM, Vol. 32, No. 1 (January 1989); ACM Press; New York, USA; 1989; S. 106-110

Wand, Y.; Wang, R.Y.: Anchoring Data Quality Dimensions in Ontological Foundations; erschienen in: Communications of the ACM, Vol. 39, No. 11 (November 1996); New York; 1996; S. 91

Wang, H.; Zaniolo, C.: Using SQL to Build New Aggregates and Extenders for Object-Relational Systems; erschienen in: Abbadi, A. E.; Brodie, M. L.;Chakravarthy, S.; Dayal, U.; Kamel, N.; Schlageter, G.; Whang, K.-Y. (Hrsg.): Proceedings of 26th International Conference on Very Large Data Bases (VLDB); September 10-14, 2000; Cairo, Ägypten

Wang, H.; Zaniolo, C.: User Defined Aggregates in Object-Relational Systems; erschienen in: 16th International Conference on Data Engineering (ICDE'2000); IEEE Computer Society 2000; San Diego, USA; 2000; S. 135-144

Wang, R.Y.; Ziad, M.; Lee, Y.W.: Data Quality; Kluwer Academic Publishers; Norwell, Massachusetts; 2001

Wen, Y.: Text Mining Using HMM and PPM, University of Waikato, Hamilton, 2001

Wilz, M.: Aspekte der Kodierung phonetischer Ähnlichkeiten in deutschen Eigennamen; Magisterarbeit; Universität Köln, Philosophische Fakultät, Institut für Linguistik, Abteilung Phonetik; Köln; 2005

Windheuser, U.: Datenqualität - Ein nicht unlösbares Problem; erschienen in: DMC Anwendertage 2004; Chemnitz

Winkler, W.E.: Quality of Very Large Databases; Research Report RR2001; U.S. Bureau of the Census; 2001

Winter, M.; Helfert, M.; Herrmann, C.: Das metadatenbasierte Datenqualitätssystem der Credit Suisse; erschienen in: von Maur, E.; Winter, R. (Hrsg.): Vom Data Warehouse zum Corporate Knowledge Center - Proceedings der Data Warehousing 2002; Physica-Verlag; Heidelberg; 2002; S. 161-171

Winter, M.; Herrmann, C.; Helfert, M.: Datenqualitätsmanagement für Data-Warehouse-Systeme - Technische und organisatorische Realisierung am Beispiel der Credit Suisse; erschienen in: von Maur, E.; Winter, R. (Hrsg.): Data Warehouse Management: Das St. Galler Konzept zur ganzheitlichen Gestaltung der Informationslogistik; Springer; Heidelberg; 2003; S. 221-240

Witt, K.U.: Algebraische Grundlagen der Informatik; 2. Auflage; Vieweg Verlag; Wiesbaden; 2005

Witten, I.H.; Frank, E.: Data Mining; Hanser Verlag; München, Wien; 2001

Witten, I.H.; Moffat, A.; Bell, T.C.: Managing Gigabytes – Compressing and Indexing Documents and Images, 2nd. ed.; Morgan Kaufmann; 1999; San Francisco

Yan, L.L.; Özsu, M. T.: Conflict Tolerant Queries in AURORA; erschienen in: Proceedings of the Fourth IECIS International Conference on Cooperative Information Systems; IEEE Computer Society; Washington, DC, USA; 1999; S. 279-290

Yoon, K.P.; Hwang, C.-L.: Multiple Attribute Decision Making; Sage Publications; Thousands Oaks, Kalifornien; 1995

Ziegler, P.; Dittrich, K.R.: Three Decades of Data Integration - All Problems solved?; erschienen in: Jacquart, R. (Eds.): 18th IFIP World Computer Congress (WCC 2004), Volume 12, Building the Information Society, August 22-27; Kluwer; Toulouse, France; S. 3-12

Zobel, J.; Dart, P.: Finding Approximate Matches in Large Lexicons; erschienen in: Software - Practice and Experience, Vol. 25, Issue 3 (March 1995); John Wiley & Sons; New York, USA; S. 331-345