Universität Kaiserslautern
Fachbereich Informatik
AG Datenbanken und Informationssysteme
Prof. Dr. Theo Härder

Ähnlichkeitssuche in objekt-relationalen Datenbanksystemen

Diplomarbeit

von

Michael P. Haustein

Betreuer:

Dipl.-Inform. Wolfgang Mahnke
Dr.-Ing. Norbert Ritter

Mai 2002

Hiermit erkläre ich an Eides statt, daß ich die vorliegende Arbeit selbständig und unter ausschließlicher Verwendung der angegebenen Literatur angefertigt habe.

Kaiserslautern, den 23. Mai 2002

Michael P. Haustein

Inhaltsverzeichnis

Kapitel 1 Einleitung 5

1.1 Motivation ... 5

1.2 Inhalt und Gliederung ... 6

Kapitel 2 Ähnlichkeitsmaße 7

2.1 Ähnlichkeit ... 7

2.2 Ähnlichkeitsmaße für Attribut-Wert Repräsentationen ... 7

2.2.1 Distanz- und Ähnlichkeitsmaße für binäre Attribute ... 7
2.2.2 Distanzmaße für numerische Attribute ... 9
2.2.3 Das Lokal-Global-Prinzip ... 10
2.2.4 Lokale Ähnlichkeitsmaße für numerische Attribute ... 10
2.2.5 Lokale Ähnlichkeitsmaße für symbolische Attribute ... 11
2.2.6 Lokale Ähnlichkeitsmaße für taxonomisch geordnete Attribute ... 11
2.2.7 Lokale Ähnlichkeitsmaße für Zeichenketten ... 12
2.2.8 Lokale Ähnlichkeitsmaße für Datums- und Zeittypen ... 15
2.2.9 Globale Ähnlichkeitsmaße ... 15
2.2.10 Gewichtsmodelle für globale Ähnlichkeitsmaße ... 16

2.3 Ähnlichkeitsmaße für objekt-orientierte Repräsentationen ... 16

2.3.1 Mehrwertige Attribute ... 17
2.3.2 Unsichere Information ... 17

2.4 Ähnlichkeitsmaße für Graph Repräsentationen ... 18

2.4.1 Ähnlichkeitsmaße basierend auf Isomorphie ... 18
2.4.2 Graph-Editing ... 19

2.5 Ähnlichkeitsmaße für prädikatenlogische Repräsentationen ... 19

2.5.1 Atomare Formeln wie binäre Attribute behandeln ... 19
2.5.2 Anwenden der Anti-Unifikation ... 20
2.5.3 Ähnlichkeit durch logische Inferenz ... 20

Kapitel 3 Vorhandene Ansätze objekt-relationaler DBMS 21

3.1 Die Ansätze der IBM DB2 ... 21

3.1.1 Indexierung ... 21
3.1.2 Modifikation der Suchanfrage ... 22
3.1.3 Text Extender ... 23
3.1.4 Net Search Extender ... 26
3.1.5 Text Information Extender ... 29

3.2 Die Ansätze von IBM Informix 31
3.2.1 Indexierung 32
3.2.2 Typen der Ähnlichkeitssuche 34
3.2.3 Anpassen der Suchergebnisse 37
3.2.4 Beispiel 39
3.3 Die Ansätze von Oracle 40
3.3.1 Architektur zur Indexierung 40
3.3.2 Indextypen 43
3.3.3 Beispiel 46
3.4 Fazit 47

Kapitel 4 Anforderungen 49
4.1 Datenmodellierung in objekt-relationalen Datenbanksystemen 49
4.2 Lokale Ähnlichkeitsmaße 50
4.3 Globale Ähnlichkeitsmaße 50
4.4 Berücksichtigung der Objekthierarchie 51
4.5 Parametrisierung 51

Kapitel 5 Realisierungsaspekte 53
5.1 Starke Parametrisierung 53
5.1.1 Eigenschaften 53
5.1.2 Nachteile 54
5.1.3 Beispiel SFB-501-Reuse-Repository 54
5.1.4 Unterstützung der Ähnlichkeitsberechnung für Taxonomien 56
5.2 Schwache Parametrisierung 57
5.2.1 Inverse Indizes 57
5.2.2 Inverser Index für Taxonomien 58
5.2.3 Inverser Index für ganzzahlige Daten 60
5.2.4 Inverser Index für Fließkommazahlen 63
5.2.5 Inverser n-gram Index für Zeichenketten 65
5.2.6 Function-value-indexing 67
5.2.7 Aggregation 68
5.2.8 NULL-Werte 69
5.3 Keine Parametrisierung 69
5.3.1 Eigenschaften 69
5.3.2 Unterstützung mit einem globalen inversen Index 70
5.3.3 Leistungsbewertung 70
5.4 Zusammenfassung 71

Kapitel 6 Zusammenfassung und Ausblick 73
6.1 Zusammenfassung 73
6.2 Ausblick 74

Anhang A Suche im SFB-501-Reuse-Repository 77

A.1 Problematik ... 77

A.2 Einsatz einer Puffertabelle ... 78

A.3 Meßergebnisse ... 78

Anhang B Leistungsmessungen mit Indizes 81

B.1 Messungen mit lokalem Index für Taxonomien ... 81

B.1.1 Datenbankschema ... 81

B.1.2 Suchanfragen ... 83

B.1.3 Meßergebnisse ... 84

B.2 Messungen mit lokalem Index für ganzzahlige Daten ... 86

B.2.1 Datenbankschema ... 86

B.2.2 Suchanfragen ... 87

B.2.3 Meßergebisse ... 88

B.3 Function-value-indexing ... 90

B.4 Messungen mit lokalem Index für Fließkommadaten ... 92

B.4.1 Datenbankschema ... 92

B.4.2 Suchanfragen ... 93

B.4.3 Meßergebisse ... 94

B.5 Messungen mit lokalem Index für Zeichenketten ... 96

B.5.1 Datenbankschema ... 96

B.5.2 Suchanfragen ... 98

B.5.3 Meßergebnisse ... 99

B.5.4 Nutzung objekt-relationaler Technologien ... 101

B.6 Messungen mit globalem Index ... 102

B.6.1 Datenbankschema ... 102

B.6.2 Suchanfragen ... 104

B.6.3 Meßergebnisse ... 105

Kapitel 7 Literaturverzeichnis 107

Kapitel 1 Einleitung

Die ähnlichkeitsbasierte Suche läßt sich in vielen Gebieten im Datenbankbereich einsetzen und gewinnt immer mehr an Bedeutung. Da zunehmend mehr objekt-relationale Datenbanksysteme eingesetzt werden ([STONE99]), bietet es sich an, eine ähnlichkeitsbasierte Suche in Verbindung mit objekt-relationalen Konzepten durchzuführen. Sie ermöglicht Suchanfragen, bei denen der Anwender die exakten Attributwerte der zu findenden Datensätze nicht kennen muß, bzw. Datensätze benötigt, die zu einer gegebenen Vergleichsinstanz (eventuell eine benutzerdefinierte Objektinstanz) möglichst passend sind.

1.1 Motivation

In zahlreichen Gebieten finden Ansätze ähnlichkeitsbasierter Suche bereits ihre Anwendung.

- Beim fallbasierten Schließen (*case-based-reasoning CBR*) wird die Ähnlichkeitssuche zum Auffinden eines möglichst ähnlichen Falles verwendet, dessen Lösungsansatz mit einer Adaption an das Ausgangsproblem zu dessen Lösung führt.
- Im Wissensmanagement wird die ähnlichkeitsbasierte Suche zum Auffinden von Wissensinformation oder Erfahrungselementen eingesetzt, die nach Vorgabe von speziellen Eigenschaften durch eine Vergleichsinstanz möglichst passend ermittelt werden müssen.
- Beim Text- oder Bild-Retrieval müssen Dokumente oder Bilder aufgefunden werden, die möglichst viele zuvor spezifizierter Eigenschaften (sogenannten *Features*) aufweisen.
- Durch die Ähnlichkeitssuche werden Anfragen auf Datenbanken möglich, die Tabellen mit vielen Attributen beinhalten, bei denen die Semantik einzelner Attribute oder Attributwerte nicht jedem Benutzer vollständig bekannt sind.
- In der Biochemie wird eine ähnlichkeitsbasierte Suche für Anfragen auf Gensequenzen benutzt. Hier geht man von der Annahme aus, daß Moleküle mit ähnlichen Strukturen auch ähnliche Eigenschaften besitzen.

In dieser Diplomarbeit untersuchen wir, welche Möglichkeiten es zur Berechnung von Ähnlichkeiten zwischen Daten gibt und wie darauf aufbauend eine Ähnlichkeitssuche durchgeführt werden kann. Wir betrachten, wie Suchanfragen in einem objekt-relationalen Datenbanksystem realisiert werden können und beschreiben Verfahren zur Verbesserung des Antwortzeitverhaltens.

Insbesondere untersuchen wir dabei, wie der Einsatz von objekt-relationalen Technologien die Möglichkeiten der Ähnlichkeitssuche beeinflußt. Objekt-relationale Datenbanksystemen stellen als Erweiterung gegenüber herkömmlichen relationalen Datenbanksystemen Möglichkeiten zur Verfügung, Tabellen in Vererbungshierarchien zu organiseren und eigene komplexe Datentypen zu definieren. Der Anwender kann eigene Funktionen entwerfen und innerhalb des Datenbanksystems ausführen. Somit können Funktionswerte für beliebige Attributwerte ohne externe Kommunikation berechnet und darauf aufbauend auch Indexierungen vorgenommen werden. Wir untersuchen, wie eine Ähnlichkeitssuche innerhalb dieser Strukuren mit benutzerdefinierten Funktionsaufrufen durchgeführt werden kann.

1.2 Inhalt und Gliederung

Im folgenden Kapitel 2 werden Ähnlichkeitsmaße (Funktionen zur Berechnung von Ähnlichkeitswerten) diskutiert. Dabei werden zahlreiche Möglichkeiten beschrieben, wie für diverse Repräsentationen von Daten ein Ähnlichkeitswert zwischen einem Datensatz und einer Vergleichsinstanz ermittelt werden kann.

Kapitel 3 beschäftigt sich mit den bereits vorhandenen Ansätzen ähnlichkeitsbasierter Suche. Da sich jedoch hinter Schlüsselwörtern wie *fuzzy* oder *similarity search* meist die Ähnlichkeitssuche auf großen Mengen indexierter Textdokumente verbirgt, werden in diesem Kapitel die Erweiterungen von Datenbanksystemen drei bekannter Hersteller zur ähnlichkeitsbasierten Textsuche vorgestellt.

In Kapitel 4 beschreiben wir Anforderungen, die wir, ausgehend von den Kapiteln 2 und 3, an ein objekt-relationales Datenbanksystem stellen, das eine ähnlichkeitsbasierte Suche auf den verwalteten Daten unterstützen soll.

Kapitel 5 behandelt einige Realisierungsaspekte der gestellten Anforderungen, die beim Entwurf der ähnlichkeitsbasierten Suchfunktionalität zu beachten sind. Dazu werden verschiedene Formen der Parametrisierung untersucht und Verfahren erläutert, durch die eine Beschleunigung der Antwortzeit erreicht werden kann. Dies wird ergänzend durch einige Leistungsmessungen auf größeren Datenmengen belegt.

In Kapitel 6 werden die Ergebnisse der einzelnen Kapitel kurz zusammengefaßt und ein Ausblick zur Thematik gegeben. Die anschließenden Anhänge A und B beschreiben genauer die in Kapitel 5 durchgeführten Leistungsmessungen.

Kapitel 2 Ähnlichkeitsmaße

Bei einer ähnlichkeitsbasierten Suche spezifiziert der Benutzer in einer Vergleichsinstanz alle ihm bekannten Attribute. Nach Auswertung der Suchanfrage erhält der Benutzer Datensätze, die der definierten Vergleichsinstanz möglichst ähnlich sind. Zusätzlich sollte der Anwender die Möglichkeit haben, die Berechnung der Ähnlichkeit zweier Objekte flexibel an seine individuellen Bedürfnisse anzupassen.

2.1 Ähnlichkeit

Als erstes stellt sich die Frage der Berechnung einer Ähnlichkeit zwischen der Vergleichsinstanz und einem Datensatz. Dazu definieren wir zunächst das Distanz- und das Ähnlichkeitsmaß.

Das *Distanzmaß* ist eine Funktion $dist : I\!D \times I\!D \to [0, 1]$ auf dem Definitionsbereich $I\!D$, die für zwei Instanzen deren Distanz berechnet. Unter der Distanz verstehen wir einen Zahlenwert, der die Summe aller Unterschiede der Instanzen ausdrückt. Dieser Zahlenwert hängt natürlich eng mit der Semantik des gewählten Distanzmaßes ab. Mit der Distanz kann man somit die Ähnlichkeit beurteilen. Dabei steht 0 für die geringste und 1 für die größte Distanz.

Unter dem *Ähnlichkeitsmaß* verstehen wir eine Funktion $sim : I\!D \times I\!D \to [0, 1]$, die für zwei Instanzen deren Ähnlichkeit berechnet; auch die Ähnlichkeit ist ein Zahlenwert zwischen 0 und 1. Dabei steht 0 für die geringste und 1 für die größte Ähnlichkeit. In die Berechnung des Ähnlichkeitsmaßes kann das Distanzmaß mit einfließen.

Klassifiziert nach der Repräsentation der Daten, betrachten wir nun im folgenden verschiedene Ansätze zur Realisierung des Distanz- und Ähnlichkeitsmaßes. Soweit nicht anders erwähnt, sind diese in [BERG01] nachzulesen.

2.2 Ähnlichkeitsmaße für Attribut-Wert Repräsentationen

Bei der Attribut-Wert Repräsentation nehmen wir an, daß die Vergleichsinstanz und der Datensatz als eine Menge von Attributen gegeben ist, d.h. wir berechnen eine Ähnlichkeit zwischen den Attributvektoren $\vec{x} = (x_1, \ldots, x_n)$ und $\vec{y} = (y_1, \ldots, y_n)$ auf dem Definitionsbereich $I\!D$.

2.2.1 Distanz- und Ähnlichkeitsmaße für binäre Attribute

Für binären Attributvektoren, d.h. $x_i, y_i \in \{0, 1\}$ werden in der Literatur zahlreiche Distanz- und Ähnlichkeitsmaße vorgestellt, von denen im folgenden einige betrachtet werden.

Der allgemein bekannte *Hamming-Abstand* beschreibt die Anzahl der sich unterscheidenden Bits zweier Binärzahlen. Durch die Multiplikation mit dem Faktor $\frac{1}{n}$ definiert man ein Distanzmaß, das proportional zu den sich unterscheidenden Bits eine Distanz zwischen 0 und 1 berechnet.

Hamming-Abstand: $dist_H(\vec{x}, \vec{y}) = \frac{1}{n} \cdot |\{i | x_i \neq y_i\}|$

Das entsprechend dem Hamming-Abstand definierte Ähnlichkeitsmaß wird als *Simple Matching Coeffizient (SMC)* bezeichnet. Dazu betrachtet man zur Berechnung eines Ähnlichkeitswertes die Anzahl der übereinstimmenden Bits beider Vektoren $\vec{x}$ und $\vec{y}$.

Simple Matching Coeffizient: $sim_{SMC}(\vec{x}, \vec{y}) = \frac{1}{n} \cdot |\{i | (x_i = y_i)\}|$

Eine Verallgemeinerung des SMC ist der *Weighted SMC (WSMC)*. Hier wird für jedes Attribut x_i zusätzlich ein Gewicht w_i definiert, das die Relevanz des entsprechenden Attributs innerhalb des Vektors ausdrückt. Damit der Ähnlichkeitswert zwischen 0 und 1 liegt, muß die Summe der Gewichte gleich eins sein ($\sum w_i = 1$). Ist diese Bedingung nicht erfüllt, müssen die definierten Gewichte vor der Berechnung normiert werden.

Weighted SMC: $sim_{WSMC}(\vec{x}, \vec{y}) = \sum_{x_i = y_i} w_i$

Eine weitere Verallgemeinerung des SMC ist der *Non-linear SMC (NLSMC)*, bei dem zusätzlich zur bereits berechneten SMC-Ähnlichkeit noch die Anzahl der jeweils übereinstimmenden und nicht übereinstimmenden Attributwerte in den Ähnlichkeitswert mit einfließt. Über den Parameter α kann das Verhalten der Funktion angepaßt werden: Für $0 < \alpha < 0,5$ wird die Anzahl der nicht übereinstimmende Attribute höher bewertet, für $0,5 < \alpha < 1$ wird die Anzahl der übereinstimmenden Attribute höher gewichtet. Für $\alpha = 0,5$ geht aus dem NLSMC wieder der SMC hervor.

Non-linear SMC: $sim_{NLSMC}(\vec{x}, \vec{y}) = \dfrac{\alpha \cdot sim_{SMC}(\vec{x}, \vec{y})}{(\alpha \cdot sim_{SMC}(\vec{x}, \vec{y})) + (1-\alpha)(1 - sim_{SMC}(\vec{x}, \vec{y}))}$

Der Ansatz von Tversky betrachtet den Bitvektor als eine Liste von Features, die vorhanden sind (1) oder nicht vorhanden sind (0). Eine Funktion *f* wertet die Indexmenge der Bitpositionen aus, an denen in der Vergleichsinstanz und im Datensatz beide oder jeweils ein Feature gesetzt ist. Das Fehlen eines Features sowohl in der Vergleichsinstanz als auch im Datensatz wird durch den Funktionswert von f berücksichtigt. Durch die Parametrisierung mittels α, β und γ wird die Gewichtung übereinstimmender und nicht übereinstimmender Features beeinflußt. Dieses Ähnlichkeitsmaß wird zum Beispiel zur Bestimmung der Ähnlichkeit von Molekülen (Darstellung von Molekülen mit Featurevektoren, Moleküle mit gemeinsamen Features haben oft auch ähnliches Verhalten) eingesetzt ([BRAD97]).

Tversky Ähnlichkeitsmaß:
$$\begin{aligned} sim_T(\vec{x}, \vec{y}) = {} & \alpha \cdot f(\{i | x_i = y_i = 1\}) - \\ & \beta \cdot f(\{i | x_i = 1 \wedge y_i = 0\}) - \\ & \gamma \cdot f(\{i | x_i = 0 \wedge y_i = 1\}) \end{aligned}$$

2.2.2 Distanzmaße für numerische Attribute

Gehen wir nun davon aus, daß die Vektoren $\vec{x}$ und $\vec{y}$ aus numerische Attributen zusammengesetzt sind; wir nehmen an $x_i, y_i \in [0, 1]$. Sollte diese Voraussetzung nicht erfüllt sein, so kann diese durch Normierung der Werte erreicht werden. Darauf aufbauend, kann man nun die *City Block Metrik*, die *Euklidische Distanz* und die *Maximum Norm* definieren.

City Block Metrik: $dist_{CBM}(\vec{x}, \vec{y}) = \frac{1}{n} \cdot \sum_{i=1}^{n} |x_i - y_i|$

Euklidische Distanz: $dist_{Euklid}(\vec{x}, \vec{y}) = \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} (x_i - y_i)^2}$

Maximum Norm: $dist_{Max}(\vec{x}, \vec{y}) = max_{i=1}^{n} |x_i - y_i|$

Diese drei Distanzmaße können durch die *Minowski Norm* verallgemeinert werden. Dabei bestimmt der Parameter p das Verhalten des Minowski Distanzmaßes: City Block Metrik für p=1, Euklidische Distanz für p=2 oder Maximum Norm für $p \to \infty$. Je größer p gewählt wird, desto stärker ist der Einfluß großer Distanzen auf die gesamte Norm.

Minowski Norm: $dist_{Minowski}(\vec{x}, \vec{y}) = \left(\frac{1}{n} \cdot \sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}}$

Auch für die Distanzmaße numerischer Attribute können Gewichte ($w_1, \ldots, w_n$ mit $\sum w_i = 1$) zur Betonung der Wichtigkeit einzelner Attribute eingeführt werden. Damit erhält man die *gewichtete City Block Metrik*, die *gewichtete Euklidische Distanz* und die *gewichtete Maximum Norm*, welche wiederum in der *gewichteten Minowski Norm* verallgemeinert werden können.

Gewichtete Minowski Norm:

$$dist_{WMinowski}(\vec{x}, \vec{y}) = \left(\sum_{i=1}^{n} w_i \cdot |x_i - y_i|^p\right)^{\frac{1}{p}}$$

Eine Erweiterung der gewichteten Euklidischen Distanz ist die *Quadratform Distanz*. Dazu wird eine Gewichtsmatrix $W = [w_{ij}]$ definiert, in der Gewichte zum Vergleich zweier Attribute eingetragen werden. Damit das Distanzmaß nur Werte zwischen 0 und 1 liefert, müssen wir voraussetzen, daß W positiv definit und $\sum_i \sum_j w_{ij} = 1$ ist.

Quadratform Distanz: $dist_{QuadForm}(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} \cdot (x_i - y_i)(x_j - y_j)}$

2.2.3 Das Lokal-Global-Prinzip

Die Berechnung der Ähnlichkeit zwischen einer Vergleichsinstanz und einem Datensatz kann übersichtlicher und benutzerspezifischer durch Anwendung des Lokal-Global-Prinzips durchgeführt werden. Dies ist vor allem dann notwendig, wenn die Attribute eines Datensatzes verschiedene Datentypen aufweisen. Dazu wird jeweils zwischen den einzelnen Attributen ein lokaler Ähnlichkeitswert berechnet. Die Ähnlichkeit zwischen der Vergleichsinstanz und einem Datensatz wird dann mittels einer Aggregatfunktion aus den lokalen Ähnlichkeitswerten gebildet.

Ein *lokales Ähnlichkeitsmaß* ist ein Ähnlichkeitsmaß für ein einzelnes Attribut A_i und wird definiert durch $sim_{A_i} : T_{i_{Range}} \times T_{i_{Range}} \to [0, 1]$, wobei $T_{i_{Range}}$ der Wertebereich des Attributs A_i ist. Durch entsprechende Parametrisierung kann das lokale Ähnlichkeitsmaß individuell angepaßt werden.

Das *globale Ähnlichkeitsmaß* wird aus den lokalen Ähnlichkeitsmaßen berechnet und wird definiert als $sim_\Phi(\vec{x}, \vec{y}) = \Phi(sim_{A_1}(x_1, y_1), \ldots, sim_{A_n}(x_n, y_n))$. Die Funktion $\Phi : [0, 1]^n \to [0, 1]$ wird als *Aggregatfunktion* bezeichnet und muß die folgenden Bedingungen erfüllen:

- $\Phi(0, \ldots, 0) = 0$
- Φ ist monoton steigend in jedem Argument

Durch geeignete Parametrisierung kann das globale Ähnlichkeitsmaß an Benutzervorgaben angepaßt werden und so beispielsweise die Relevanz einzelner Attribute mittels eines Gewichtsmodells berücksichtigen.

Im folgenden betrachten wir lokale Ähnlichkeitsmaße für verschiedene Datentypen, einige globale Ähnlichkeitsmaße und Gewichtsmodelle.

2.2.4 Lokale Ähnlichkeitsmaße für numerische Attribute

Der Vorteil numerischer Attribute besteht darin, daß auf ihnen bereits eine Ordnung definiert ist und somit recht einfach eine Distanz berechnet werden kann. Sei im folgenden δ eine Funktion, die die Distanz zweier numerischer Werte berechnet.

Lineare Differenz: $\delta(x, y) = x - y$

Logarithmische Differenz: $\delta(x, y) = \ln(x) - \ln(y)$ für $x, y \in IR^+$
$\delta(x, y) = -\ln(-x) + \ln(-y)$ für $x, y \in IR^-$

Die logarithmische Differenz eignet sich typischerweise in physikalischen Bereichen, wenn zum Beispiel die Ähnlichkeit zwischen 1V und 1,2V gleich der Ähnlichkeit zwischen 1mV und 1,2mV sein soll.

Basierend auf einer entsprechend gewählten Distanzfunktion δ kann nun die *symmetrische* oder *asymmetrische Ähnlichkeit* definiert werden.

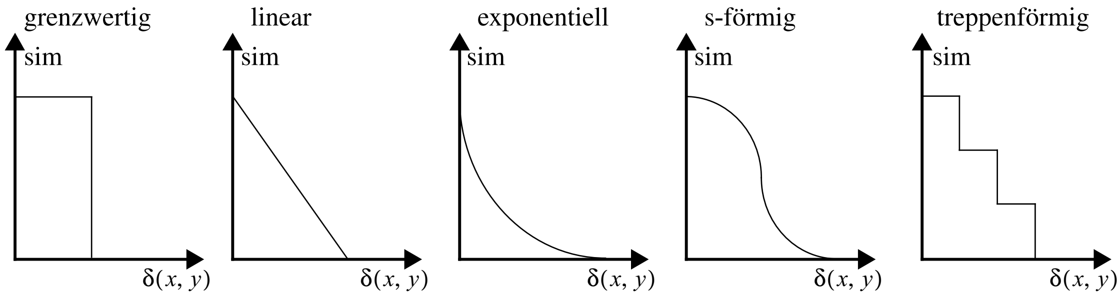
Symmetrische Ähnlichkeit: $sim_{A_i}(x, y) = f(|\delta(x, y)|)$

Asymmetrische Ähnlichkeit:

$$sim_{A_i}(x, y) = \begin{cases} f(\delta(x, y)) & \text{für } x > y \\ 1 & \text{für } x = y \\ g(\delta(x, y)) & \text{für } x < y \end{cases}$$

Die Funktionen f und g sind monoton fallend und bilden $I\!R$ auf [0,1] ab, so daß f(0)=1 und g(0)=1 sind. Sie beschreiben damit das Abnehmen der Ähnlichkeit mit zunehmender Distanz. Abbildung 1 zeigt mehrere Möglichkeiten, die lokale Ähnlichkeit für numerische Attribute zu berechnen.

Abbildung 1 Lokale Ähnlichkeitsmaße für numerische Attribute



2.2.5 Lokale Ähnlichkeitsmaße für symbolische Attribute

Unter dem Begriff symbolische Attribute faßt man Attribute mit einem Wertebereich $T_{A_i} = \{v_1, \ldots, v_n\}$ zusammen. Das Attribut kann nur einen der Werte v_1 bis v_n annehmen. Wir unterscheiden zwischen ungeordneten und geordneten symbolischen Attributen.

Bei den **ungeordneten symbolischen Attributen** geht man davon aus, daß auf den einzelnen Werten der Menge T_{A_i} keine Ordnung besteht.

Um dafür ein Ähnlichkeitsmaß einzuführen, definiert man eine $n \times n$-Matrix $S = [s_{ij}]$ mit $0 \le s_{ij} \le 1$, in der die Ähnlichkeitswerte zwischen den Elementen von T_{A_i} abgelegt werden. Die Berechnung der lokalen Ähnlichkeit zweier Attribute mit Werten v_x und v_y erfolgt nach dem *tabellarischen Ähnlichkeitsmaß* wie folgt:

Tabellarisches Ähnlichkeitsmaß: $sim_{A_i}(v_x, v_y) = s_{xy}$

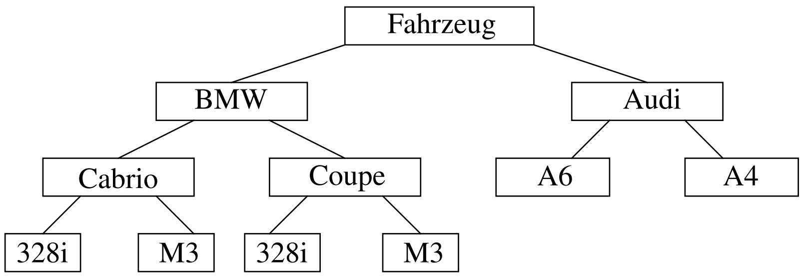
Diese Methode ist jedoch nur anwendbar, wenn die Menge T_{A_i} eine überschaubare Anzahl von Elementen aufweist, da die Größe der Matrix S und somit die Zahl der zu verwaltenden Einträge quadratisch mit der Kardinalität von T_{A_i} wächst.

Bei **geordneten symbolischen Attributen** nehmen wir an, daß auf den Elementen der Menge T_{A_i} eine totale Ordnung besteht, d.h. für die Elemente der Menge $T_{A_i} = \{v_1, \ldots, v_n\}$ gilt $v_i < v_{i+1}$ für $i = 1, \ldots, n-1$. Wendet man nun die schon behandelten lokalen Ähnlichkeitsmaße für numerische Attribute auf die Indizes der Werte v_i an, so kann man damit ein Ähnlichkeitsmaß für geordnete symbolische Attribute einführen.

2.2.6 Lokale Ähnlichkeitsmaße für taxonomisch geordnete Attribute

Ein Spezialfall der geordneten symbolischen Atrribute sind die sogenannten Taxonomien. Eine Taxonomie ist ein Baum, bei dem jeder Knoten für einen möglichen Attributwert steht. Durch die Baumstruktur bestehen nun im Gegensatz zur einfachen Listendarstellung der Menge T_{A_i} zusätzliche Beziehungen zwischen den Attributwerten, die zur Berechnung der Ähnlichkeit herangezogen werden können. Dabei beschreiben die Blattknoten des Baums konkrete Objekte, die inneren Baumknoten Klassen von Objekten, die zur Strukturierung verwendet werden. Abbildung 2 zeigt das Beispiel einer Taxonomie von Fahrzeugen, die nach Hersteller und Form organisiert sind.

Abbildung 2 Beispiel einer Taxonomie von Fahrzeugen



Um ein geeignetes Ähnlichkeitsmaß zu definieren, führen wir zwei Notationen auf der Baumstruktur ein. Für zwei Knoten K_1 und K_2 bedeutet $K_1 > K_2$, daß K_2 Nachfolger von K_1 ist, d.h. K_1 liegt auf dem Pfad von K_2 zur Wurzel. Weiterhin steht $<K_1,K_2>$ für den nächsten gemeinsamen Vorgängerknoten von K_1 und K_2.

Als Bedingung für ein Ähnlichkeitsmaß ergibt sich $sim(K, K_1) \le sim(K, K_2)$, wenn $\langle K, K_1 \rangle > \langle K, K_2 \rangle$, d.h. die Ähnlichkeit von K und K_1 ist kleiner gleich der Ähnlichkeit von K und K_2, wenn der nächste gemeinsame Vorgänger von K und K_1 näher an der Wurzel liegt als der nächste gemeinsame Vorgänger von K und K_2. Im Beispiel betrachtet bedeutet dies anschaulich, daß die Ähnlichkeit eines *M3 Cabrio* und eines *328i Cabrio* größer ist als die Ähnlichkeit eines *M3 Cabrio* und eines *328i Coupe*, da deren nächster gemeinsamer Vorgänger *BMW* näher an der Wurzel liegt als der Knoten *Cabrio*.

Um einen konkreten Zahlenwert für die Ähnlichkeit zweier Blattknoten zu bestimmen, wird jedem inneren Knoten K_i des Baums ein Ähnlichkeitswert $s_i \in [0, 1]$ zugeordnet, derart, daß $s_i \le s_j$ wenn $K_i > K_j$, d.h. die Ähnlichkeitswerte nehmen im Baum in Richtung der Wurzel ab. Damit stellt der Ähnlichkeitswert eines inneren Knotens eine untere Schranke für die Ähnlichkeit zwei beliebiger Nachfolgerknoten dar. Um die Ähnlichkeit zweier Blattknoten zu ermitteln, wird der nächste gemeinsame Vorgängerknoten bestimmt und dessen Ähnlichkeitswert als Ähnlichkeit der Blattknoten übernommen. Je näher dieser Vorgänger an den Blattknoten liegt, bzw. je weiter er von der Wurzel entfernt ist, desto mehr Eigenschaften haben beide Blattknoten gemeinsam und desto ähnlicher sind sie sich.

Ähnlichkeitsmaß für Taxonomie-Blattknoten:

$$sim(K_1, K_2) = \begin{cases} 1 & \text{wenn } K_1 = K_2 \\ s_{\langle K_1, K_2 \rangle} & \text{wenn } K_1 \ne K_2 \end{cases}$$

2.2.7 Lokale Ähnlichkeitsmaße für Zeichenketten

Um einen Ähnlichkeitswert von zwei Zeichenketten (Strings) zu berechnen, wird der Attributwert der Vergleichsinstanz (String1) und des Datensatzes (String2) in Großbuchstabendarstellung gebracht und darauf die jeweiligen Verfahren angewendet. Im folgenden betrachten wir kurz die *Substring-Methode*, *Edit-Distanz*, *n-Grams* und das *SoundEx*-Verfahren.

Substring-Methode. Die Substring-Methode ist ein sehr einfaches und nicht besonders effektives Verfahren, um zwei Zeichenketten als ähnlich zu werten. Nachdem beide Strings in Großbuchstabendarstellung gebracht worden sind, wird überprüft, ob der Attributwert der Vergleichsinstanz ein Teilstring des Attributwertes des Datensatzes ist (Ähnlichkeitswert 1) oder ob diese Eigenschaft nicht vorliegt (Ähnlichkeitswert 0).

Substring-Methode: $sim_{substring}(string_1, string_2) = \begin{cases} 1 \text{ wenn string}_1 \text{ Substring von string}_2 \\ 0 \text{ sonst} \end{cases}$

Edit-Distanz. Beim Verfahren der Edit-Distanz nach wird versucht, String1 mit den Operationen Einfügen, Löschen und Ersetzen in String2 zu überführen. Die Zuordnung von Zeichen des ersten Strings zu Zeichen des zweiten Strings wird als *Alignment* bezeichnet. Jeder der Operationen werden über ein entsprechendes Modell Kosten zugeordnet. Nach [MERK01] sind beide Strings in Großbuchstabendarstellung gegeben, allerdings ist dies keine Voraussetzung für das Verfahren, da auch Kosten für die Umwandlung von Klein- in Großbuchstaben (und umgekehrt) definiert werden können. Die Ähnlichkeit der beiden Zeichenketten wird nach der Überführung aus den ermittelten Gesamtkosten berechnet.

Es erweist sich als großes Problem, das Alignment mit den geringsten Kosten zu ermitteln, da es sich hierbei um ein NP-vollständiges Problem handelt. Es gibt jedoch zahlreiche Algorithmen der Bioinformatik, die durch Approximation ein recht akzeptables Ergebis mit einer Komplexität unter O(n^2) erreichen (z. B. FastA oder BLAST [MERK01]).

Das *Damerau-Levenshtein Maß* ([DELTA95]) ordnet den Operationen Einfügen und Löschen feste Kosten zu, die Kosten der Ersetzung werden durch eine Funktion bestimmt, die beispielsweise den Abstand der Zeichen auf der Tastatur zur Bewertung von Tippfehlern ermittelt.

n-Grams. Als n-Grams werden Teilzeichenketten der Länge n eines gegebenen Strings bezeichnet ([DELTA95], [KART02]). Die Ähnlichkeit zweier Strings wird aus der Anzahl der gemeinsamen n-Grams und der Zahl der insgesamt vorhandenen n-Grams berechnet. Üblicherweise werden dazu digrams (n=2) oder trigrams (n=3) verwendet. Abbildung 3 zeigt ein Beispiel nach [DELTA95] für eine Ähnlichkeitsberechnung zwischen dem Suchbegriff RECEIEVE und den Wörtern RECEIVE und RECEIVER mit Verwendung von trigrams.

Abbildung 3 Ähnlichkeitsberechnung mit trigrams

trigrams	**Suchbegriff**	**Vergleichswörter**	
	RECEIEVE	RECEIVE	RECEIVER
REC	✔	✔	✔
ECE	✔	✔	✔
CEI	✔	✔	✔
EIE	✔	✘	✘
IEV	✔	✘	✘
EVE	✔	✘	✘
EIV	✘	✔	✔
IVE	✘	✔	✔
VER	✘	✘	✔
Ähnlichkeitswert		**3 / 8**	**3 / 9**

Der Suchbegriff RECEIEVE besitzt sechs trigrams: REC, ECE, CEI, EIE, IEV und EVE. Das Vergleichswort RECEIVE besitzt die trigrams REC, ECE, CEI, EIV und IVE; das Vergleichswort RECEIVER die trigrams REC, ECE, CEI, EIV, IVE und VER. RECEIEVE und RECEIVE weisen insgesamt acht verschiedene trigrams auf, davon die drei gemeinsamen REC, ECE und CEI. Die übrigen trigrams treten jeweils nur in einem der beiden Wörter auf. RECEIEVE und RECEIVER besitzen zusammen neun verschiedene trigram, davon treten ebenfalls die drei trigram REC, ECE und CEI in beiden gemeinsam auf. Somit ergeben sich die Ähnlichkeitswerte $\frac{3}{8}$ und $\frac{3}{9}$, das Wort RECEIVE wird somit als ähnlicher gewertet.

SoundEx. Der SoundEx Algorithmus wurde von Magaret Odell und Robert Russell Anfang dieses Jahrhunderts entwickelt und in den 20er Jahren patentiert ([SCHEI01]). Er basiert auf der Annahme, daß Wörter, die ähnlich klingen, auch semantisch ähnlich zu beurteilen sind. Jedes Wort wird in einen Buchstaben und eine dreistellige Ziffernfolge kodiert, Wörter mit der gleichen Kodierung werden als ähnlich gewertet (binäres Ähnlichkeitsmaß).

SoundEx: $sim_{SoundEx}(string_1, string_2) = \begin{cases} 1 \text{ wenn } SoundEx(string_1) = SoundEx(string_2) \\ 0 \text{ sonst} \end{cases}$

Nach [NARA00] wird zunächst das zu kodierende Wort in eine Folge von Großbuchstaben umgewandelt. Der Anfangsbuchstabe des Wortes wird als Buchstabe für die Kodierung verwendet. Danach werden die Buchstaben A, E, I, O, U, H, W und Y gestrichen. Den nun verbleibenden Buchstaben wird gemäß der folgenden Tabelle in Abbildung 4 eine Ziffer zugeordnet, die in den Kode übernommen wird.

Abbildung 4 Kodeziffern für die Buchstabengruppen im SoundEx Algorithmus

1	B, F, P, V
2	C, G, J, K, Q, S, X, Z
3	D, T
4	L
5	M, N
6	R

Dabei ist zu beachten, daß Buchstaben, deren Kodeziffer dieselbe wie die des Vorgängers ist, ebenfalls gestrichen und nicht in den Kode übernommen werden. Nach drei ermittelten Kodeziffern bricht der Algorithmus ab, die restlichen Buchstaben werden nicht mehr berücksichtigt. Ist das Wort vollständig abgearbeitet, und es wurden dabei noch keine drei Kodeziffern ermittelt, wird der Kode mit 0-Ziffern aufgefüllt. Abbildung 5 zeigt die Berechnung des SoundEx Kodes für die Wörter Schmidt und Smith, beide Wörter werden als ähnlich gewertet.

Abbildung 5 Berechnung eines SoundEx Kodes

Wort	Streichen von A, E, I, O, U, H, W, Y	Streichen der Buchstaben, deren Vorgänger die gleiche Kodeziffer hat	Berechneter SoundEx Kode
SCHMIDT	SCMDT	SMD	S530
SMITH	SMT	SMT	S530

2.2.8 Lokale Ähnlichkeitsmaße für Datums- und Zeittypen

Für Attribute des Typs Datum oder Zeit können zur Berechnung eines Ähnlichkeitswertes die Verfahren für Ähnlichkeitsmaße auf numerischen Attributen herangezogen werden.

Dazu wird die Differenz zwischen zwei Daten bzw. Zeiten berechnet, d.h. man erhält einen definierten Abstand beispielsweise in Tagen bzw. Minuten. Aus diesem Abstand läßt sich nun mit einem gewählten Ähnlichkeitsmaß für numerische Attribute eine Ähnlichkeit zwischen den beiden Werten errechnen.

2.2.9 Globale Ähnlichkeitsmaße

Globale Ähnlichkeitsmaße definieren sich durch die Anwendung einer Aggregatfunktion Φ auf die lokalen Ähnlichkeitswerte. Die Aggregatfunktion Φ besteht aus einer Basis-Aggregatfunktion (z.B. Durchschnittsbildung) und einem Gewichtsmodell, das für die lokalen Ähnlichkeiten $s_1, \ldots, s_n$ durch die Gewichte $w_1, \ldots, w_n$ ($\sum_i w_i = 1$) Relevanzen festlegt.

Die *gewichtete Durchschnittsaggregation* ist die am häufigsten gewählte Aggregation. Jedes Attribute trägt zur globalen Ähnlichkeit entsprechend seines zugewiesenen Gewichts bei.

Gewichtete Durchschnittsaggregation: $\Phi(s_1, \ldots, s_n) = \sum_{i=1}^{n} (w_i \cdot s_i)$

Die *Maximum Aggregation* bestimmt die Gesamtähnlichkeit durch die größte lokale Ähnlichkeit und berechnet damit eine Art disjunktive globale Ähnlichkeit. Wenn ein einzelnes Attribut eine hohe Ähnlichkeit aufweist, so wird daraus auf eine hohe Gesamtähnlichkeit geschlossen.

Maximum Aggregation: $\Phi(s_1, \ldots, s_n) = max_{i=1}^{n}(w_i \cdot s_i)$

Die *Minimum Aggregation* bestimmt die Gesamtähnlichkeit durch die kleinste lokale Ähnlichkeit und berechnet damit eine Art konjunktive globale Ähnlichkeit. Wenn ein einzelnes Attribut eine geringe Ähnlichkeit aufweist, so wird daraus auf eine geringe Gesamtähnlichkeit geschlossen.

Minimum Aggregation: $\Phi(s_1, \ldots, s_n) = min_{i=1}^{n}(w_i \cdot s_i)$

Die *Minowski Aggregation* ist eine Verallgemeinerung der gewichteten Aggregation. Je größer der Parameter p gewählt wird, desto höher ist der Einfluß des Attributs mit der größten lokalen Ähnlichkeit auf die Gesamtähnlichkeit. Für $p \to \infty$ verhält sich die Minowski Aggregation wie die Maximum Aggregation.

Minowski Aggregation: $\Phi(s_1, \ldots, s_n) = \left(\sum_{i=1}^{n} w_i \cdot s_i^p\right)^{\frac{1}{p}}$

Die *k-Maximum Aggregation* ist eine Verallgemeinerung der Maximum Aggregation. Die Gesamtähnlichkeit wird von der k-größten lokalen Ähnlichkeit bestimmt, d.h. wenn k von n lokalen Ähnlichkeitswerten hoch sind, wird daraus auf eine hohe globale Ähnlichkeit geschlossen.

k-Maximum Aggregation: $\Phi(s_1, \ldots, s_n) = w_{i_k} \cdot s_{i_k}$ mit $w_{i_r} \cdot s_{i_r} \geq w_{i_{r+1}} \cdot s_{i_{r+1}}$

Die *k-Minimum Aggregation* ist eine Verallgemeinerung der Minimum Aggregation. Die Gesamtähnlichkeit wird von der k-kleinsten lokalen Ähnlichkeit bestimmt, d.h. wenn k von n lokalen Ähnlichkeitswerten niedrig sind, wird daraus auf eine geringe globale Ähnlichkeit geschlossen.

k-Minimum Aggregation: $\Phi(s_1, \ldots, s_n) = w_{i_k} \cdot s_{i_k}$ mit $w_{i_r} \cdot s_{i_r} \leq w_{i_{r+1}} \cdot s_{i_{r+1}}$

2.2.10 Gewichtsmodelle für globale Ähnlichkeitsmaße

Jede der beschriebenen Aggregatfunktionen ist mit einem Gewichtsvektor $\vec{w}$ parametrisiert. Über die damit definierten Gewichte wird die Relevanz jedes einzelnen Attributs innerhalb des Datensatzes angegeben. Die Gewichte lassen sich je nach Bezug in verschiedene Gewichtsmodelle einteilen.

Globale Gewichte. Das einfachste Gewichtsmodell benutzt einen globalen Gewichtsvektor für alle Anfragen. Diese Gewichte sind nach domänenspezifischen Gesichtspunkten ausgewählt und da sie statisch sind, können sie auch als Teil der Aggregatfunktion betrachtet werden.

Klassenspezifische Gewichte. Für jede Klasse wird ein eigener Gewichtsvektor festgelegt. Die Relevanz der Attribute wird damit der Klassensemantik angepaßt.

Datensatzspezifische Gewichte. Für jeden Datensatz wird ein eigener Gewichtsvektor angegeben. Das kann sinnvoll sein, wenn nicht jeder Attributwert auf sicherer Information beruht. Attribute mit möglicherweise falschen Werten können somit gleich beim Abspeichern des Datensatzes als weniger aussagekräftig markiert werden.

Benutzerspezifische Gewichte. Jeder Benutzer kann direkt beim Spezifizieren einer Vergleichsinstanz die Gewichtung der Attribute wählen und damit das Suchergebnis nach seinen individuellen Bedürfnissen beeinflussen.

2.3 Ähnlichkeitsmaße für objekt-orientierte Repräsentationen

Zur Ermittlung des Ähnlichkeitsmaßes von objekt-orientierten Repräsentationen wendet man das in Abschnitt 2.2.3 behandelte Lokal-Global-Prinzip auf objekt-orientierter Ebene an. Es wird die Ähnlichkeit zwischen einer Vergleichsinstanz und der Instanz einer gegebenen Klassenhierarchie berechnet. Dazu wird mittels lokaler Ähnlichkeitsmaße die Ähnlichkeit der gemeinsamen Klassenattribute berechnet, diese Ähnlichkeiten wiederum mit einer Aggregatfunktion zur Gesamtähnlichkeit zusammengefaßt. Diese Gesamtähnlichkeit bezeichnet man als *Intra-Klassen-Ähnlichkeit*.

Allerdings wird dabei die Struktur der Klassenhierarchie nur bedingt berücksichtigt (lediglich durch die gemeinsamen Attribute). Daher wendet man ein dem Prinzip der Taxonomien aus Abschnitt 2.2.6 sehr ähnliches Verfahren an. Jeder Klasse wird ein Ähnlichkeitswert zwischen 0 und 1 zugeordnet, der die maximale Ähnlichkeit aller Subelemente angibt. Auch dieser Ähnlichkeitswert muß wie bei den Taxonomien von der Wurzel der Klassenhierarchie ausgehend immer größer werden. Für die Berechnung der Ähnlichkeit zweier Instanzen wird nun die nächste gemeinsame Oberklasse bestimmt und deren Ähnlichkeitswert übernommen. Die so errechnete Ähnlichkeit wird als *Inter-Klassen-Ähnlichkeit* bezeichnet.

Die Objekt-Ähnlichkeit zweier Objekte o_1 und o_2 errechnet sich damit aus der Intra- und der Inter-Klassen-Ähnlichkeit gemäß folgender Formel.

Objekt-Ähnlichkeit: $sim(o_1, o_2) = sim_{intra}(o_1, o_2) \cdot sim_{inter}(o_1, o_2)$

2.3.1 Mehrwertige Attribute

Ein mehrwertiges Attribut a^d (z.B. Mengen) eines Datensatzes kann mehrere Werte einer vorgegebenen Wertemenge annehmen. Ebenso kann der Benuzter in der Vergleichsinstanz a^s mehrere Werte aus dieser Menge spezifizieren. Sei also $a^d = \{v_1, \ldots, v_n\}$ und $a^s = \{u_1, \ldots, u_m\}$, dann kann die Attributähnlichkeit mit einer Aggregation der lokalen Ähnlichkeiten aller paarweise kombinierten Elemente von a^d und a^s berechnet werden.

Globale Ähnlichkeit: $$sim(a^s, a^d) = \Phi\left(\begin{bmatrix} sim_{Lokal}(u_1, v_1) & \ldots & sim_{Lokal}(u_1, v_m) \\ \ldots & \ldots & \ldots \\ sim_{Lokal}(u_n, v_1) & \ldots & sim_{Lokal}(u_n, v_m) \end{bmatrix}\right)$$

Je nach der Semantik, die hinter der Spezifikation von Werten bei a^s steht, unterscheiden wir zwischen *Any-Value* Semantik und *Multi-Property* Semantik.

Any-Value Semantik. Der Benutzer gibt in der Vergleichsinstanz mehrere Werte für ein Attribut an und erwartet, daß das Attribut in den Ergebisdatensätzen mindestens einen dieser Werte enthält. Daraus ergibt sich für die Attributähnlichkeit die Anwendung der Maximum-Aggregation.

Ähnlichkeit bei Any-Value Semantik: $sim(a^s, a^d) = \Phi((s_{ij})) = max\ s_{ij}$

Multi-Property Semantik. Der Benutzer gibt in der Vergleichsinstanz mehrere Werte für ein Attribute an und erwartet, daß das Attribut in den Ergebnisdatensätzen möglichst alle diese Werte enthält. Dazu muß eine Funktion γ gefunden werden, die eine Abbildung der Elemente $\{u_1, \ldots, u_n\}$ auf die Elemente $\{v_1, \ldots, v_m\}$ vornimmt. Diese kann je nach Semantik ohne Einschränkung, injektiv, surjektiv oder bijektiv sein.

Ähnlichkeit bei Multi-Property Semantik: $sim(a^s, a^d) = \Phi((s_{ij})) = \frac{1}{n} \cdot max\left\{\sum_{i=1}^{n} s_{i\gamma(i)}\right\}$

2.3.2 Unsichere Information

Ein weiterer Grund, mehrwertige Attribute zu verwenden, ist das Vorkommen unsicherer Information. Beim Einlagern des Datensatzes ist der Attributwert nicht genau bekannt, daher werden alle zu diesem Zeitpunkt für sinnvoll erachteten Werte angegeben. Je nach Problemstellung kann die Attributähnlichkeit *pessimistisch*, *optimistisch* oder *durchschnittlich* durch entsprechende Aggregation bestimmt werden.

Unsichere Information, pessimistisch: $sim(a^s, a^d) = \Phi((s_{ij})) = min\ s_{ij}$

Unsichere Information, optimistisch: $sim(a^s, a^d) = \Phi((s_{ij})) = max\ s_{ij}$

Unsichere Information, durchschnittlich: $sim(a^s, a^d) = \Phi((s_{ij})) = \frac{1}{n \cdot m} \cdot \sum_{i,j} s_{ij}$

2.4 Ähnlichkeitsmaße für Graph Repräsentationen

Wir nehmen nun an, daß die in der Datenbank gespeicherte Information eines Datensatzes einen Graphen repräsentiert, d.h. wir suchen Funktionen zur Berechnung der Ähnlichkeit zweier gegebener Graphen (die Vergleichsinstanz und den Datensatz).

2.4.1 Ähnlichkeitsmaße basierend auf Isomorphie

Wir definieren zunächst den Begriff der Isomorphie. Zwei Graphen x und y, jeweils bestehend aus einer Menge von Knoten und Kanten, heißen i*somorph* ($x \cong y$), wenn es eine bijektive Abbildungen zwischen den beiden Knotenmengen gibt.

Damit können wir nun ein recht einfaches Ähnlichkeitsmaß basierend auf der Graphenisomorphie definieren.

Ähnlichkeitsmaß basierend auf Graphenisomorphie: $sim(x, y) = \begin{cases} 1 \text{ wenn } x \cong y \\ 0 \text{ sonst} \end{cases}$

Einen Schritt weiter geht die Definition der Teilgraph-Isomorphie. Zwei Graphen x und y heißen *teilgraph-isomorph* ($x \ll y$), wenn es einen Teilgraph y' von y gibt, derart daß $x \cong y'$.

Die Teilgraph-Isomorphie erlaubt die Definition nicht-symmetrischer Ähnlichkeitsmaße, die auf der Teilgrapheigenschaft beruht.

Ähnlichkeitsmaße basierend auf Teilgraph-Isomorphie: $sim_1(x, y) = \begin{cases} 1 \text{ wenn } x \ll y \\ 0 \text{ sonst} \end{cases}$

$$sim_2(x, y) = \begin{cases} 1 \text{ wenn } y \ll x \\ 0 \text{ sonst} \end{cases}$$

Diese beiden binären Ähnlichkeitsmaße (Ähnlichkeitswert strikt 0 oder 1) lassen sich durch die Definition des größten gemeinsamen Teilgraphen zu nicht-binären Ähnlichkeitsmaßen erweitern.

Es ist z der *größte gemeinsame Teilgraph* zweier Graphen x und y ($z = lcsg(x, y)$), wenn $z \ll x$ und $z \ll y$ und es zudem keinen größeren (d.h. mehr Knoten beinhaltenden) Teilgraph von x und y gibt.

Somit kann man mit Hilfe der Größe des größten gemeinsamen Teilgraphen ein symmetrisches nicht-binäres Ähnlichkeitsmaß einführen.

Symmetisches Ähnlichkeitsmaß basierend auf lcsg: $sim(x, y) = f\left(1 - \frac{|lcsg(x, y)|}{max\{|x|, |y|\}}\right)$

Analog zu den beiden Ähnlichkeitsmaßen der Teilgraph-Isomorphie lassen sich auch hier noch zwei nicht-symmetrische Ähnlichkeitsmaße definieren.

Nicht-symmetrische Ähnlichkeitsmaße basierend auf lcsg: $sim_1(x, y) = f\left(1 - \frac{|lcsg(x, y)|}{|x|}\right)$

$$sim_2(x, y) = f\left(1 - \frac{|lcsg(x, y)|}{|y|}\right)$$

Die Realisierbarbeit solcher Verfahren, die auf Graphenisomorphie beruhen, erweist sich jedoch als sehr schwierig. Zwar handelt es sich nach [GROHE99] nachweisbar nicht um NP-vollständige Probleme, allerdings ist auch noch kein Algorithmus bekannt, der solche Probleme in polynomialer Zeit löst. Es existiert zwar nach [BERG01] eine Methode von Babai und Kucera (1979), die die Isomorphie zweier Graphen in O(n) berechnet, allerdings funktioniert diese nicht für beliebige Graphen, sondern verlangt nach gewissen Einschränkungen.

2.4.2 Graph-Editing

Ein Verfahren ähnlich der Edit-Distanz von Zeichenketten beschreibt das Graph-Editing. Soll die Ähnlichkeit zwischen zwei Graphen x und y berechnet werden, so versucht man, Graph x in Graph y zu überführen. Dazu gibt es die Operationen Knoten/Kante einfügen, Knoten/Kante löschen und Knoten/Kante umbenennen, denen jeweils entsprechende Kosten mittels einem Kostenmodell zugewiesen sind. Durch die benötigten Gesamtkosten der Überführung von Graph x in Graph y läßt sich damit ein konkreter Ähnlichkeitswert berechnen.

Jedoch stößt auch hier die Anwendung des Verfahrens schnell an die Grenzen der Realisierbarkeit, da das Auffinden der Operationsfolge mit den geringsten Gesamtkosten ein NP-vollständiges Problem darstellt.

2.5 Ähnlichkeitsmaße für prädikatenlogische Repräsentationen

Besteht ein Datensatz aus einer Menge von atomaren Formeln, so liegt dieser in einer prädikatenlogischen Repräsentation vor. Im folgenden werden drei Ansätze beschrieben, um auf solchen Repräsentationen Ähnlichkeitsmaße zu definieren.

2.5.1 Atomare Formeln wie binäre Attribute behandeln

Zunächst besteht die Möglichkeit, die beiden Mengen von atomaren Formeln $x = \{\varphi_1, \ldots, \varphi_n\}$ und $y = \{\psi_1, \ldots, \psi_m\}$ als binäre Vektoren in der Dimension $|x \cup y|$ zu betrachten. Die Ähnlichkeit wird aufgrund der Atome bestimmt, die sowohl in x als auch in y identisch auftreten. Dazu werden zwei binäre Vektoren der Dimension $|x \cup y|$ gebildet, deren i-te Komponente jeweils auf 1 gesetzt wird, falls das entsprechende i-te Atom von $x \cup y$ in x bzw. y vorkommt. Auf die somit gebildeten binären Vektoren können die in Kapitel 2.2.1 bereits definierten Ähnlichkeitsmaße übertragen werden.

Die so eingeführten Ähnlichkeitsmaße sind sehr einfach und erlauben keinerlei Berücksichtigung der Semantik der betrachteten Formeln.

2.5.2 Anwenden der Anti-Unifikation

Durch das Benutzen der Anti-Unifikation können wir Ähnlichkeitsmaße in vergleichbarer Weise wie beim Anwenden der Teilgraph-Isomorphie in Kapitel 2.4.1 einführen. Dazu definieren wir zunächst die Anti-Unifikation.

Gegeben seien zwei atomare Formeln φ und ψ. Wir nennen φ *allgemeiner* als ψ ($\varphi \geq \psi$), genau dann wenn eine Substitution σ existiert, derart daß $\psi = \sigma(\varphi)$. Die *Anti-Unifikation* von zwei Formeln φ und ψ ist eine Formel χ ($\chi = au(\varphi, \psi)$), mit den Eigenschaften $\chi \geq \varphi$, $\chi \geq \psi$ und es existiert keine Formel χ', so daß $\chi' \geq \varphi$, $\chi' \geq \psi$ und $\chi' < \chi$.

Die Größe der Anti-Unifikation kann nun wiederum benutzt werden, um ein Ähnlichkeitsmaß zu definieren. Betrachten wir die Größe einer atomaren Formel $|\varphi|$ als die Anzahl der Symbole und Konstanten, so kann das folgende symmetrische Ähnlichkeitsmaß definiert werden.

Ähnlichkeitsmaß basierend auf Anti-Unifikation: $sim(x, y) = f\left(1 - \frac{|au(x, y)|}{max\{|x|, |y|\}}\right)$

Ebenso wie in Kapitel 2.4.1 können auch die beiden nicht-symmetrischen auf Anti-Unifikation basierenden Ähnlichkeitsmaße definiert werden.

Nicht-symmetrische Ähnlichkeitsmaße: $sim_1(x, y) = f\left(1 - \frac{|au(x, y)|}{|x|}\right)$

$$sim_2(x, y) = f\left(1 - \frac{|au(x, y)|}{|y|}\right)$$

Besteht ein einzelner Datensatz aus mehreren Formeln, so kann dies wie ein mehrwertiges Attribut in objekt-orientierten Repräsentationen betrachtet (siehe Kapitel 2.3.1) und die oben definierte Ähnlichkeit als lokales Ähnlichkeitsmaß behandelt werden. Im Gegensatz zu den Ähnlichkeitsmaßen in Graph-Repräsentationen können die Ähnlichkeiten der prädikatenlogischen Repräsentationen in polynomialer Zeit berechnet werden, weil die Darstellung einer prädikatenlogischen Formel einem Baum (und nicht einem beliebigen Graphen) entspricht.

2.5.3 Ähnlichkeit durch logische Inferenz

Ein weiterer Ansatz, ein binäres Ähnlichkeitsmaß zu definieren, ist das Verwenden einer logischen Theorie Σ, in der alle Bedingungen der Anwendungsdomäne festgelegt sind. Dabei werden alle Formeln als ähnlich betrachtet, deren Implikation in der entsprechend gegebenen Theorie gefolgert werden können.

Ähnlichkeitsmaß basierend auf einer logischen Theorie Σ: $sim(x, y) = \begin{cases} 1 \text{ wenn } & \Sigma \models x \rightarrow y \\ 0 \text{ sonst} & \end{cases}$

Kapitel 3 Vorhandene Ansätze objekt-relationaler DBMS

Die zur Zeit verfügbaren objekt-relationalen Datenbanksysteme unterstützen kaum ähnlichkeitsbasiertes Suchen. Einige Ansätze sind in der Bildverarbeitung durch einfache Mustererkennung oder Auffinden von Farbverteilungen vorhanden. Hauptsächlich aber verbergen sich hinter Begriffen wie *fuzzy search* die Suche in großen Mengen von Textdokumenten durch Auffinden von ähnlich buchstabierten oder klingenden Wörtern oder das Expandieren der Anfrage unter Zuhilfenahme von Thesauri.

In den folgenden Kapiteln 3.1 bis 3.3 werden die Ansätze der Datenbanksysteme IBM DB2, IBM Informix und Oracle kurz vorgestellt und in einem Fazit in Kapitel 3.4 die dort verfügbaren Funktionalitäten beurteilt.

3.1 Die Ansätze der IBM DB2

IBM stellt zur Erweiterung des Datenbanksystems DB2 für die Ähnlichkeitssuche auf Textdokumenten drei sogenannte Extender zur Verfügung. Diese erweitern das Datenbanksystem nach der Installation um diverse Datentypen und/oder Funktionalitäten. Nach einer kurzen Betrachtung der Arbeitsschritte, die ein Dokument während der Indexierung durchläuft und die zur Modifikation der Suchanfrage herangezogen werden, gehen wir danach auf den *Text Extender*, *Net Search Extender* und *Text Information Extender* näher ein.

3.1.1 Indexierung

Um auf großen Textmengen effizient suchen zu können, müssen diese indexiert werden. Die komplette Suche verläuft danach über die erstellten Indexdaten. Die Qualität des Suchergebnisses ist dabei in entscheidendem Maße von der Qualität der Indexdaten abhängig. Nach [DB2TEAP00] wird bei der Erstellung der Indexdaten wie folgt vorgegangen.

Für eine Tabelle, die Textdaten enthält, auf denen später einmal gesucht werden soll, müssen Indizes angelegt werden. Dabei kann ein Index für die gesamte Tabelle oder für jede Spalte ein separater Index definiert werden. Bei der Definition eines Indexes muß der Typ der Suchfunktion (z. B. präzise, unscharf oder linguistisch), die später auf diese Spalte angewendet wird, mit angegeben werden. Daher können zur Unterstützung verschiedener Suchanfragen auf einer Spalte für diese auch mehrere Indizes angelegt werden. Beim Anlegen eines Indexes auf einer Tabellenspalte wird automatisch eine zusätzliche 60-Byte-VARCHAR Spalte erzeugt (sog. *Handle*), die einige Metadaten (Dokument-ID, Sprache, Indexname) verwaltet. Danach beginnt die Text-Suchmaschine mit dem Aufbau der Indexdaten.

Termerkennung. Zunächst müssen die innerhalb des Dokuments vorhandenen Terme identifiziert werden. Das ist bei einzelnen Wörtern recht einfach, bei Zahlenkolonnen mit Punkt und Komma, denen beispielsweise noch ein Währungszeichen vorangestellt ist oder folgt, muß auf länderspezifische Regelungen Rücksicht genommen werden.

Normalisierung. Dann werden die erkannten Terme normalisiert. Dazu werden Großbuchstaben durch Kleinbuchstaben und Umlaute durch deren Schreibweise ae, ue, oe, usw. ersetzt. Dies ermöglicht bei den folgenden Schritten eine einfachere Verarbeitung.

Satzerkennung. Zum Erkennen von Sätzen werden die Positionen von Satzzeichen wie Punkt, Doppelpunkt, Semikolon, Ausrufezeichen und Fragezeichen untersucht. Mit Hilfe einer sprachenspezifischen Liste wird ausgeschlossen, daß es sich bei dem erkannten Zeichen um das Ende einer Abkürzung handelt.

Dekomposition. Der nächste Schritt ist die Dekomposition von Wörtern. In germanischen Sprachen wie z.B. Deutsch kommen viele zusammengesetzte Wörter vor. Diese werden wiederum mit Hilfe eines Wörterbuchs in einzelne Teile zerlegt und diese zur Indexierung herangezogen.

Stopwort-Filterung. Bei der Stopwort-Filterung werden Wörter aus dem Dokument entfernt, die nicht zur Indexierung beitragen sollen (z.B. mein, für, um, zu, so, usw.). Dazu wird eine Stopwortliste verwaltet, in der alle auszuschließenden Wörter aufgelistet sind.

Grundform-Bildung. Zuletzt werden die Wörter des Dokuments auf ihre Grundform zurückgeführt. Beispielsweise wird das Wort *Mäuse* als *maus* indexiert. Dadurch findet man bei einer späteren Suchanfrage alle Dokumente zum Thema *Maus*, unabhängig davon, in welcher Form das Wort *Maus* in einem der Dokumente vorhanden ist.

3.1.2 Modifikation der Suchanfrage

Um mit den in der Suchanfrage spezifizierten Wörtern ein möglichst gutes Suchergebis zu erreichen, wird auch die Suchanfrage vor dem Indexzugriff modifiziert. Dazu stehen nach [DB2TEAP00] die beiden Basisoperationen Reduktion und Expansion zur Verfügung.

Bei der Reduktion werden die Wörter der Suchanfrage den in Abschnitt 3.1.1 beschriebenen Verfahren Termerkennung, Normalisierung und Stopwort-Filterung unterzogen, um somit die Suchanfrage auf deren wesentliche Bestandteile zu reduzieren.

Bei der Expansion wird die Suchanfrage durch zusätzlich generierte Bedingungen erweitert, die mit der ODER-Verknüpfung an die gestellte Anfrage angehängt werden. Wir betrachten die Synonym-, die Thesaurus- und die Termexpansion.

Synonymexpansion. Synonyme werden für jede Sprache in ihrer Grundform in einer dafür vorgesehenen Tabelle verwaltet. Vor dem Nachschlagen eines Wortes in der Synonymliste muß auch dieses in Grundform gebracht werden. Die so ermittelten Synonyme werden in der Suchanfrage ergänzt.

Thesaurusexpansion. Für die Thesaurusexpansion wird ein Vokabular von Wörtern verwaltet, die miteinander in Beziehung stehen. Ist die Thesaurusexpansion aktiviert, so werden bei der Suche auch Dokumente zurückgeliefert, die aufgrund des Thesaurus in semantischem Zusammenhang mit der Suchanfrage stehen, obwohl die in der Suchanfrage angegebenen Wörter eventuell gar nicht direkt in den Dokumenten vorhanden sind.

Termexpansion. Die Termexpansion ist die Umkehroperation zur Reduktion auf die Grundform. Da aber alle Wörter im Index bereits auf ihre Grundform reduziert worden sind, dient die Termexpansion der Hervorhebung aller passender Terme innerhalb eines Dokuments bei dessen Betrachtung.

3.1.3 Text Extender

Der Text Extender ist derzeit das Standardprodukt, um die DB2 für die Ähnlichkeitssuche auf Texten zu erweitern. Alle oben dargestellten Verfahren zur Indexierung und Modifikation der Suchanfrage werden vom Text Extender nach [DB2FTSP01] und [DB2TEAP00] vollständig angeboten.

3.1.3.1 Suchfunktionen

Die Textsuchfunktionen des Text Extenders sind in SQL integriert und können zusammen mit den von SQL bekannten Bedingungen in der WHERE-Klausel verwendet werden.

Die boole'sche Suche erlaubt die UND-, ODER- und NICHT-Verknüpfung von Wörtern oder ganzen Sätzen in der Suchanfrage.

Es besteht die Möglichkeit, in der Suchanfrage zu formulieren, daß eine Menge angegebener Wörter sich innerhalb des gleichen Paragraphen oder Satzes befinden muß (*proximity search*, *Nachbarschaftssuche*).

Die unscharfe Suche erlaubt es, Wörter aufzufinden, die ähnlich wie das spezifizierte Suchwort buchstabiert werden. Dazu muß ein Index für die unscharfe Suche angelegt werden, der mit der in Abschnitt 2.2.7 beschriebenen n-gram Technik einen ähnlichkeitsbasierten Vergleich erlaubt. Dabei werden für westliche Sprachen trigrams und asiatische Sprachen bigrams verwendet, eine Anpassung durch Parameter ist dem Benutzer nicht möglich.

Weiterhin kann ein liguistischer Index erzeugt werden, der zur Zeit 22 Sprachen unterstützt. Hier werden in einer Art Wörterbuch Wortgrundformen, Synonyme, phonetische Ergänzungen und ein Thesaurus zu den Wörtern verwaltet, so daß auch Begriffe mit ähnlicher Bedeutung oder ähnlichem Klang gefunden werden können.

Es ist möglich innerhalb einzelner Abschnitte von strukturierten Dokumenten zu suchen, dabei werden HTML, XML oder benutzerdefinierte Strukturen unterstützt. Unter benutzterdefinierten Strukturen werden ASCII-Texte mit HTML-ähnlichen Strukturierungselementen verstanden, deren einheitliche Struktur mittels eines Dokumentenmodells dem System bekannt gemacht werden (*user defined tagging*).

Die Suche ist auf einer oder mehrerer Tabellenspalten eines beliebigen Zeichenkettentyps, auf Spalten des Typs CLOB und BLOB (binäre Objekte) nach den oben aufgeführten Methoden möglich. Desweiteren kann auch auf externen Dokumenten, die mit dem DataLink Verfahren (auch als Extender verfügbar) referenziert werden, eine Suchanfrage gestartet werden. Hier unterstützt der Text Extender die Dateiformate ASCII, HTML, XML, AmiPro-Architektur, Microsoft Word, Microsoft Rich Text und WordPerfect.

Zwar kann auf beliebigen Attributtypen gesucht werden, jedoch werden deren Werte stets als Text interpretiert, und der Nutzen dieser Anwendung auf Nicht-Text-Attribute ist im Rahmen einer ähnlichkeitsbasierten Suche daher eher zweifelhaft.

Um beurteilen zu können, wie ähnlich ein Dokument der Vergleichsinstanz ist, gibt es die Möglichkeit, für das Dokument einen Bewertungsfaktor berechnen zu lassen (*ranking*). Diese skalare Funktion heißt beim Text Extender *rank()*; mit der SQL-Klausel *ORDER BY* können die gefundenen Dokumente somit nach ihrer Relevanz sortiert ausgegeben werden.

3.1.3.2 Beispiel

Betrachten wir nun ein kleines Beispiel nach [DB2FTSP01] zur Anwendung des Text Extenders. Zunächst wird die Tabelle *books* in der Datenbank *sample* angelegt.

Syntax 1 Anlegen einer Tabelle

```
CREATE TABLE books
(
  author VARCHAR(30),
  story  LONG VARCHAR,
  year   INTEGER
);
```

Nach dem Füllen der Tabelle mit Daten wird ein Index erzeugt, der die linguistische Suche auf dem Attribut *story* unterstützt; dazu soll die Tabelle um das Handle-Attribut *story_handle* erweitert werden. Das Anlegen eines Index erfolgt über den *ENABLE*-Befehl, der über ein Kommandozeilenprogramm direkt an den Text Extender geschickt wird.

Syntax 2 Indexdefinition

```
ENABLE DATABASE sample;
ENABLE TEXT COLUMN books story HANDLE story_handle
  INDEX TYPE linguistic;
```

Die Suche nach Bücherinhalten mit dem Wort *mouse* und dem Erscheinungsjahr *2000* oder später wird mittels einer herkömmlichen SQL-Anfrage durchgeführt, das Ansprechen des Text Extenders erfolgt durch den Aufruf der Funktion *DB2TX.CONTAINS* in der Where-Klausel.

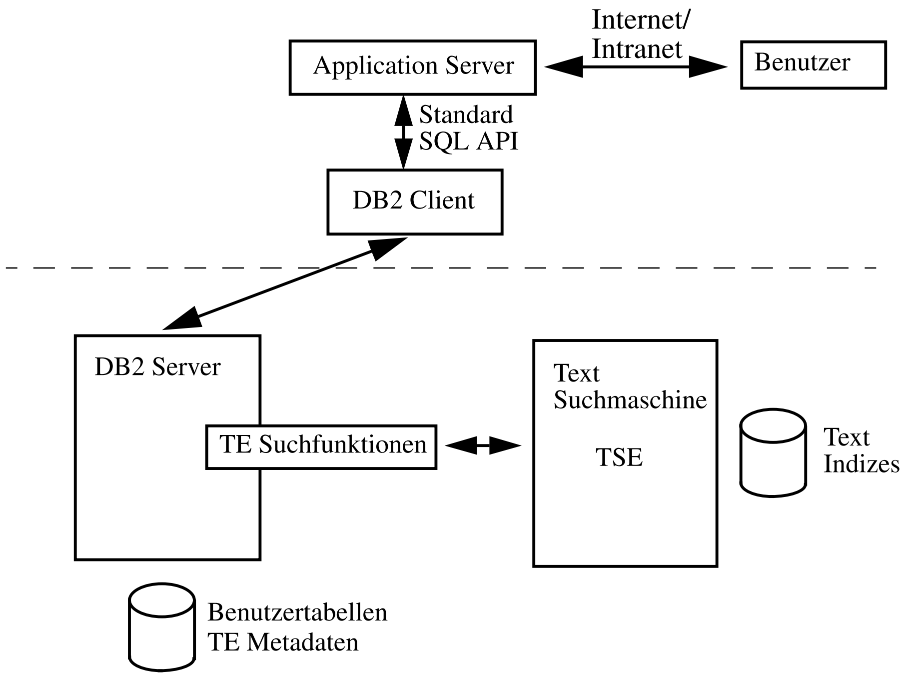
Syntax 3 Suchanfrage

```
SELECT author, story
FROM   books
WHERE  DB2TX.CONTAINS (story_handle, '\"mouse\"') = 1
  AND  year >= 2000
```

3.1.3.3 Architektur

Die folgende Abbildung 6 zeigt die Systemarchitektur beim Einsatz des DB2 Text Extenders.

Abbildung 6 Text Extender Architektur



Der Benutzer kommuniziert über einen Application-Server und den DB2-Client mit dem DB2-Server. Bei einer Suchanfrage wird eine Suchfunktion im Server aufgerufen, diese kommuniziert mit der Text-Suchmaschine.

Der Text Extender benutzt die Text-Suchmaschine TSR von IBM, diese stellt die oben beschriebene Funktionalität zur Verfügung. Laut IBM Entwicklungsabteilung kann die Text-Suchmaschine prinzipiell auch durch ein beliebig anderes „Produkt von der Stange" ersetzt werden. Der Text-Suchmaschinen-Prozeß muß nicht auf dem System des DB2-Servers laufen, das Kommunikationsprotokoll ist frei wählbar. In der Standardkonfiguration läuft die Text-Suchmaschine auf demselben System wie der DB2-Server, die Kommunikation erfolgt über Shared-Memory. Die im DB2-Server aufgerufenen Suchfunktionen des Text Extenders laufen im Modus *fenced* (getrennter Adreßraum) oder *not fenced* (im Adreßraum des DB2-Servers). Die zuletzt genannte Methode bietet zwar Leistungsvorteile, dies geschieht jedoch auf Kosten der Stabilität des DB2-Servers im Fehlerfall.

Der DB2-Server verwaltet die Benutzertabellen mit den eingelagerten Textdokumenten (oder Verweise auf externe Daten) und die Metadaten der Text-Suchmaschine (Synonyme, Stopwortliste, Thesaurus, Handles, usw.). Die Indexdaten werden von der Text-Suchmaschine selbst verwaltet und direkt im Dateisystem abgelegt. Dies bringt (laut IBM Entwicklungsabteilung) große Leistungsvorteile gegenüber einer Verwaltung der Indexdaten im Datenbankserver.

3.1.3.4 Vor- und Nachteile

Positiv zu bewerten ist die Integration der Suchfunktionen in die Anfragesprache SQL. Diese API ist vielen Anwendern bekannt und für viele Entwicklungsplattformen vorhanden. Zudem ist es recht einfach möglich, Suchergebnisse mit weiteren SQL-Prädikaten zu kombinieren.

Der Text Extender übernimmt selbständig die Aufgabe, die Daten des DB2-Servers mit den Indexdaten zu synchonisieren. So wird der Index konsistent gehalten, und die Suche liefert immer gültige Resultate.

Text Extender ist voll in die vom DB2-Server vorgegebene Authorisierung eingebunden und macht es somit überflüssig, eine eigene redundante Authorisierungskomponente implementieren zu müssen.

Nachteilig beurteilt IBM nach [DB2FTSP01], daß der Text Extender den heutigen gewaltigen Datenmengen von High-End-Anwendungen nicht mehr gewachsen ist und die Leistungsgrenze des Systems damit erreicht wird (hier wird nach [DB2FTSP01] der Net Search Extender empfohlen).

Da die Text-Suchmaschine als eigenständige Komponente realisiert ist und der DB2-Server außer einigen Verwaltungsdaten keinerlei Information über die erzeugten Indizes besitzt, kommt der DB2-Optimierer fast kaum zum Einsatz. Es liegen keine Informationen über die Kosten einer Text-Extender-Suchanfrage, die Größe des Ergebnisses oder den effizientesten Algorithmus zur Kombination von Text-Extender-Anfragen und konventionellen SQL-Prädikaten vor.

Einige weitere Nachteile ergeben sich auch bei der Installation des Text Extenders, die laut [DB2FTSP01] detaillierte DB2 Kenntnisse erfordert; darauf gehen wir hier jedoch nicht näher ein.

3.1.4 Net Search Extender

Um auch auf den oben schon angesprochenen heutigen sehr großen Datenmenge effizient suchen zu können, bietet IBM den Net Search Extender an. Zwar warnt IBM selbst davor ([DB2FTSP01]), die gewaltigen Hardwareanforderungen zu beachten, doch erhält der Kunde dafür ein System mit Anwortzeiten unter einer Sekunde, keinem Leistungsabfall bei bis zu 1000 parallelen Anfragen pro Sekunde und einer Indexierungsgeschwindigkeit von über einem Gigabyte pro Stunde.

Diese Leistung erhält der Kunde aber nur (wie schon erwähnt) durch den Einsatz sehr umfangreicher Hardwareausstattung und stark vermindertem Funktionsumfang.

3.1.4.1 Suchfunktionen

Die gesamte Suchfunktionalität des Net Search Extenders ist nicht in die SQL-Sprache integriert, sondern wird nur durch *Stored Procedures* zur Verfügung gestellt.

Boole'sche Suche, unscharfe Suche und Suche innerhalb angegebener Teile von benutzerdefiniert strukturierten Dokumenten ist wie beim Text Extender möglich.

Zur Ausgabe der Dokumente in der Reihenfolge einer berechneten Relevanz (ranking) bietet der Net Search Extender eine eigens dafür vorgesehene Funktion an, die die gefundenen Dokumente in der entsprechenden Sortierung ausgibt.

Allerdings muß der Benutzer des Net Search Extenders auf die Indexierung von externen Dokumenten mittels DataLink, XML- und HTML-Unterstützung, den linguistischen Index, den Thesaurus und die Nachbarschaftssuche verzichten.

3.1.4.2 Beispiel

Sehen wir uns nach [DB2FTSP01] ein kleines Syntaxbeispiel zur Anwendung des Net Search Extenders an. Zunächst wird die Tabelle *products* in der Datenbank *sample* zur Aufbewahrung der Daten angelegt.

Syntax 4 Anlegen einer Tabelle

```
CREATE TABLE products
(
  article_no  INTEGER NOT NULL,
  name        VARCHAR(100),
  description LONG VARCHAR,
  price       INTEGER,
  PRIMARY KEY (article_no)
)
```

Nachdem die Tabelle *products* mit Daten gefüllt worden ist, erzeugen wir einen Textindex auf der Spalte *description*. Dies geschieht mit dem *ENABLE TEXT COLUMN* Befehl, der über ein Kommandozeilenprogramm direkt an den Net Search Extender geschickt wird.

Syntax 5 Indexdefinition

```
ENABLE DATABASE sample;
ENABLE TEXT COLUMN products description
  INDEX prod_dsc
  USING article_no
  OPTIMIZE ON (description, price)
  ORDER BY price ASC
  DATABASE sample;
```

Durch die Indexdefinition wird der Index *prod_dsc* angelegt, dessen Daten nach dem Preis vorsortiert werden. Den größten Leistungsgewinn gegenüber den anderen Extendern erzielt der Net Search Extender durch die Möglichkeit, komplette Indexdaten im Hauptspeicher des Systems zu halten. Durch die Angabe *OPTIMIZE ON (description, price)* wird der Extender angewiesen, die Indexdaten für *description* und *price* im Hauptspeicher zu verwalten.

Nun kann die Suchanfrage von Java-, DB2 CLI- oder IBM Net.Data-Applikationen aus gestartet werden.

Syntax 6 Suchanfrage

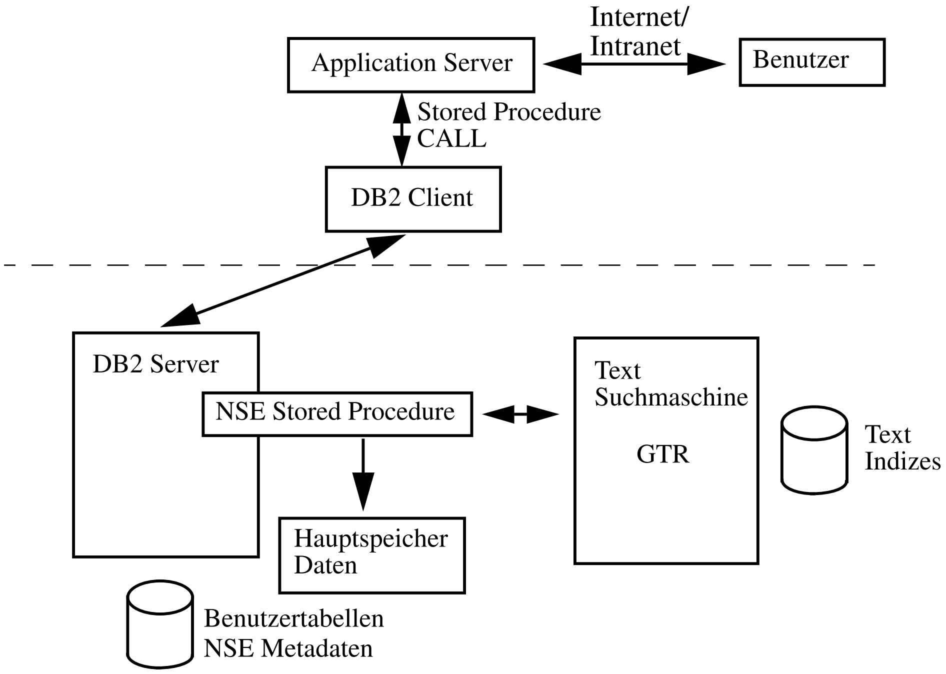
```
CALL textSearch (?, '\"blue\"', 0, 0, 0, 100, 'PROD_DSC',
                 '$DB2NX_INSTOWNERHOMEDIR/db2nx/indices', null, null, 0, ?)
```

Der Aufruf der *textSearch* Funktion sucht nun auf dem Index *prod_dsc* nach Produktbeschreibungen, die das Wort *blue* enthalten. Auf eine genauere Erklärung aller Parameter der Funktion wird in diesem Zusammenhang verzichtet, in [DB2FTSP01] wird auf das Net Search Extender Handbuch verwiesen.

3.1.4.3 Architektur

Die folgende Abbildung 7 beschreibt die Architektur des Net Search Extenders. Obwohl diese der Architektur des Text Extenders auf den ersten Blick sehr ähnlich ist, gibt es doch einige wichtige Unterschiede.

Abbildung 7 Net Search Extender Architektur



Der Benutzer hat nur die Möglichkeit, mit einem Stored Procedure Call auf den Net Search Extender zuzugreifen. Zur Laufzeit wird damit (bis auf den Funktionsaufruf) der DB2-Server vollständig umgangen.

Auch hier wird eine in einem getrennten Prozeß laufende Text-Suchmaschine verwendet, die ihre eigenen Indexdaten aus Leistungsgründen direkt im Dateisystem verwaltet. Der Net Search Extender benutzt die Text-Suchmaschine GTR, die (wie oben schon angedeutet) bessere Leistung bietet, dafür aber weniger Funktionalität zur Verfügung stellt.

Zur Auswertung der Suchanfrage benutzt der Net Search Extender die im Hauptspeicher gehaltenen Indexdaten (möglicherweise schon geeignet vorsortiert) und erreicht somit eine hohe Leistung.

3.1.4.4 Vor- und Nachteile

Vorteile ergeben sich beim Einsatz des Net Search Extenders in erster Linie durch eine große Leistungszunahme gegenüber dem Text Extender und dem Text Information Extender. Antwortzeiten unter einer Sekunde ohne Leistungseinbruch bei bis zu 1000 parallelen Anfragen pro Sekunde und Indexierung von bis zum einem Gigabyte Datenvolumen pro Stunde machen den Net Search Extender zu einem High-End-Produkt, das vornehmlich im Web-Umfeld seinen Einsatz findet.

Erkauft wird diese Leistung durch gewaltige Hardwareanforderungen und einen recht stark eingeschränkten Funktionsumfang (vor allem keine linguistische Unterstützung), der die ähnlichkeitsbasierte Suche erst interessant machen würde.

Ergänzend kommt noch die Einschränkung hinzu, daß die Suchfunktionalität nur über eine *Stored Procedure* angeboten wird, was zur Folge hat, daß die Funktionalität des Net Search Extenders nicht mit der Mächtigkeit der Sprache SQL verknüpft werden kann.

3.1.5 Text Information Extender

Der Text Extender bietet zwar einen großen Funktionsumfang an, hat jedoch aufgrund der Text-Suchmaschine TSE und der Auskopplung vom DB2-Server als reine DB2-Applikation einige Leistungsengpässe. Daher hat sich IBM entschlossen, ein Nachfolgeprodukt, den Text Information Extender, zu entwickeln.

3.1.5.1 Suchfunktionen

Der Text Information Extender bietet zur Suche und Indexierung bis auf die Unterstützung durch einen linguistischen Index alle Möglichkeiten des Text Extenders an. Das bedeutet Indexierung externer Dateien mittels DataLink, Indexierung von binären Objekten, Unterstützung von XML, HTML und benutzerspezifisch strukturierter Dokumente, Thesaurus Unterstützung und Nachbarschaftssuche. Ebenso sind die Suchfunktionen in die Sprache SQL integriert und können mit beliebigen Prädikaten kombiniert werden.

Das Ranking von Dokumenten wird auch beim Text Information Extender unterstützt. Die dazu (ähnlich wie beim Text Extender) bereitgestellte skalare Funktion heißt *score()*.

In der aktuellen Implementierung ist es allerdings nur möglich, Indizes auf Tabellen anzulegen, deren Primärschlüssel aus einer einzelnen Spalte besteht.

3.1.5.2 Beispiel

Nach [DB2FTSP01] betrachten wir wieder ein kleines Anwendungsbeispiel zur Funktionsweise des Text Information Extenders. Zunächst wird die Tabelle *books* in der Datenbank *sample* angelegt.

Syntax 7 Anlegen einer Tabelle

```
CREATE TABLE books
(
  author VARCHAR(30),
  story  LONG VARCHAR,
  year   INTEGER
);
```

Nachdem wir Daten in die Tabelle eingefügt haben, wird ein Index zur Unterstützung der Suche angelegt. Dazu führen wir die Befehle *ENABLE DATABASE* und *CREATE INDEX* aus, die über ein Kommandozeilenprogramm direkt an den Text Information Extender geschickt werden.

Syntax 8 Indexdefinition

```
ENABLE DATABASE FOR TEXT CONNECT TO sample;
CREATE INDEX myTextIndex ON books(story) FOR TEXT CONNECT TO sample;
```

Nun können wir die Suchanfrage mit einer einfachen SQL Anweisung starten.

Syntax 9 Suchanfrage

```
SELECT author, story
FROM   books
WHERE  CONTAINS (story, '\"cat\"') = 1
  AND  year >= 2000
```

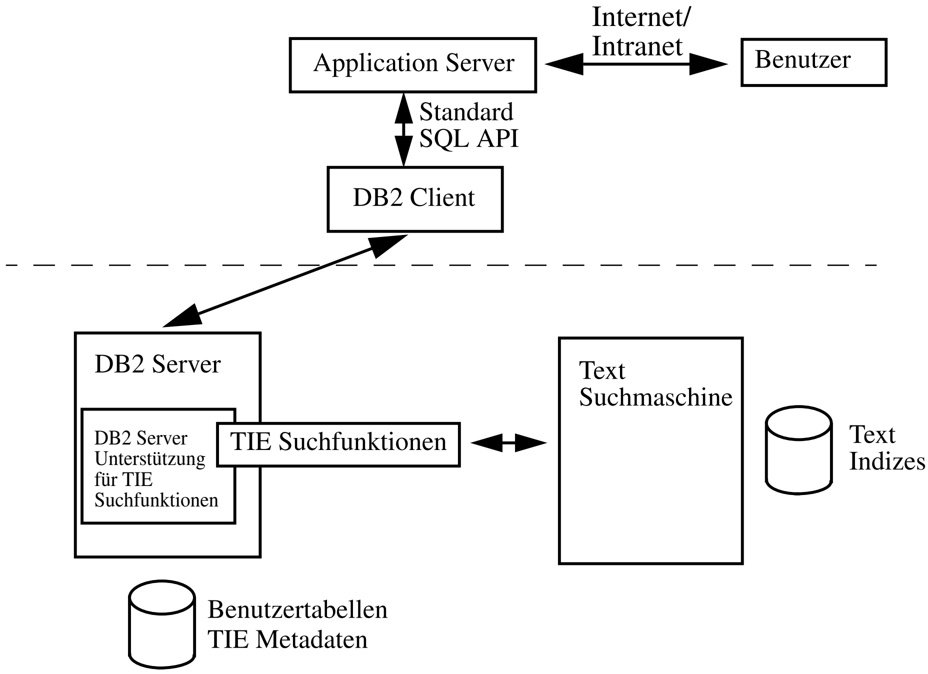
Diese Anfrage liefert nun alle Bücher über Katzen, die im Jahr 2000 oder später erschienen sind.

3.1.5.3 Architektur

Die Architektur des Text Information Extenders ist ähnlich der des Text Extenders. Signifikanter Unterschied ist die Unterstützung der Suchfunktionen des Text Information Extenders durch den DB2-Server. Somit ist es dem DB2-Optimierer möglich, höhere Leistung des Systems durch Anfrageoptimierung zu erzielen, da er die Kosten für die Ausführung einer Suchanfrage abschätzen kann.

Zur Auswertung der Suchanfrage wurde von IBM eine neue Text-Suchmaschine entwickelt, deren Kern auf der GTR Maschine basiert. Zusätzlich werden strukturierte Dokumente, Thesaurus und Nachbarschaftssuche unterstützt; die neue Maschine läuft performanter als die TSE.

Abbildung 8 Text Information Extender Architektur



3.1.5.4 Vor- und Nachteile

Viele Vorteile des Text Information Extenders können durch die Ähnlichkeit zum Text Extender übernommen werden. So sind die Suchfunktionen in die Standardsprache SQL integriert, Indexdaten werden systemseitig konsistent gehalten, und der Extender ist in die Authorisierung des DB2-Servers integriert.

Zusätzlich ist nun der Optimierer an die Fähigkeiten der neuen Text-Suchmaschine angepaßt worden und liefert durch die nun mögliche Anfrageoptimierung eine höhere Systemleistung.

Momentane Nachteile sind die Definition von Indizes nur auf Tabellen mit einfachem Primärschlüssel und die fehlende linguistische Unterstützung. Sobald diese Möglichkeiten in einer der nächsten Versionen verfügbar sind, wird der Text Information Extender laut [DB2FTSP01] den Text Extender ablösen.

3.2 Die Ansätze von IBM Informix

Für das Datenbanksystem *Informix Dynamic Server* bietet IBM das *Excalibur Text Search DataBlade* für die Ähnlichkeitssuche auf Textdokumenten an. Wie auch bei den drei Extendern für die DB2 müssen die Dokumente, auf denen gesucht werden soll, in einer Tabellenspalte gespeichert werden. Es werden die Datentypen BLOB, CLOB, LVARCHAR, CHAR und VARCHAR unterstützt.

Da auch hier vor der Suche zunächst ein Index angelegt werden muß, betrachten wir im folgenden die Möglichkeiten, die dem Anwender bei der Indexierung zur Verfügung gestellt werden.

Danach gehen wir auf die verschiedenen angebotenen Typen der Ähnlichkeitssuche ein und diskutieren das Einschränken der Suchergebnisse.

3.2.1 Indexierung

Um die Ähnlichkeitssuche zu ermöglichen, muß auf jeder Textspalte einer Tabelle, auf der später gesucht werden soll, ein Index angelegt werden. Beim Anlegen des Index können dabei verschiedene Parameter angegeben werden, die das Leistungsverhalten beeinflussen.

Der Parameter *FILTER* (mögliche Werte *NONE*, *STOP_ON_ERROR* oder *CONTINUE_ON_ERROR*) veranlaßt, daß Dokumente beim Erstellen der Indexdaten einem Filter unterzogen werden, und nur die Dokumentdaten, die diesen Filter passieren, mit in die Indexdaten aufgenommen werden. Filter existieren beispielsweise für Microsoft Word, Rich Text, HTML und zahlreiche andere Format, so daß nach Anwendung eines Filters nur die reine Textinformation eines Dokuments (und nicht die Formatierungsdaten) zur Indexierung herangezogen wird.

Der Parameter *WORD_SUPPORT* (mögliche Werte *EXACT* oder *PATTERN*) bestimmt die Methode, die zum Vergleich zweier Wörter benutzt wird. *EXACT* erlaubt nur ein binäres Ähnlichkeitsmaß (die Wörter sind genau gleich oder nicht). *PATTERN* ermöglicht die Berechnung eines abgestuften Ähnlichkeitsmaßes zwischen 0 und 100 (genaueres dazu in Abschnitt 3.2.2.1 zur Mustersuche).

Mit dem Parameter *PHRASE_SUPPORT* (mögliche Werte *NONE*, *MEDIUM* oder *MAXIMUM*) wird die Suche nach Wortmengen unterstützt, d.h. eine vorgegeben Menge von Worten soll möglichst in der gegebenen Reihenfolge im Dokument auftreten. Der Vorgabewert *NONE* erlaubt keine Wortmengenanfrage mit diesem Index, *MEDIUM* und *MAXIMUM* erlauben Anfragen dieser Art, liefern jedoch unterschiedlich präzise Ergebnisse. *MEDIUM* liefert im Gegensatz zu *MAXIMUM* eventuell auch Dokumente zurück, die das Suchkriterium nicht erfüllen, dafür wird jedoch eine Leistungssteigerung und geringerer Speicherplatzbedarf für die Indexdaten versprochen.

Mit den Parametern *STOPWORD_LIST*, *MATCH_SYNOMYM* und *CHAR_SET* können individuelle Stopwortisten, Synonymlisten und CharacterSets zur Suche herangezogen werden. Dies wird in den folgenden Abschnitten genauer diskutiert.

3.2.1.1 Stopwortlisten

Die Stopwortliste beinhaltet eine Menge von Wörtern, die bei der Analyse des Dokuments nicht zur Indexierung herangezogen werden. Bei der Installation des Excalibur Text Search DataBlades wird eine Vorgabeliste in englischer Sprache eingerichtet, diese kann aber beliebig erweitert werden. Es ist ebenfalls möglich, gleichzeitig mehrere Stopwortlisten (beispielsweise sprachenspezifisch) zu verwalten.

Stopwortlisten werden als einfache Textdateien im Dateisystem angelegt. Innerhalb dieser Textdatei werden die Stopwörter fortlaufend ohne Formatierungsinformation aufgelistet und mit den vordefinierten Funktionen *etx_CreateStopWlst()* und *etx_DropStopWlst* in das Datenbanksystem geladen oder wieder entfernt. Mit dem Parameter *STOPWORD_LIST=<NAME>* kann bei der Definition des Index eine individuelle, zuvor erstellte Stopwortliste mit dem Namen *<NAME>* verwendet werden.

Das folgende Beispiel nach [IFMXEX01] zeigt das Registrieren einer Stopwortliste aus der Datei */local0/excal/stp_word* unter dem Namen *my_stopwordlist* im Datenbank-Speicherbereich *sbsp1*.

Syntax 10 Registrieren einer Stopwortliste

```
EXECUTE PROCEDURE etx_CreateStopWlst
  ('my_stopwordlist', '/local0/excal/stp_word', 'sbsp1')
```

3.2.1.2 Synonymlisten

Die Synomymliste verwaltet eine Menge synomymer Wörter. Auch hier ist das gleichzeitige Vorhandensein mehrerer Listen im System möglich, die Synonymlisten werden jedoch erst bei der Suchanfrage benötigt.

Benutzerdefinierte Synonymlisten können ebenfalls als Textdateien im Dateisystem erzeugt werden. Dazu werden in einer Datei alle synonymen Wörter durch Leerzeichen getrennt hintereinander in einer Zeile vermerkt, mittels den Funktionen *etx_CreateSynWlst()* und *etx_DropSynWlst()* in das Datenbanksystem geladen oder wieder entfernt. Die folgende Syntax 11 zeigt nach [IFMXEX01] den typischen Inhalt einer Synonymliste.

Syntax 11 Synonymlistendatei

```
clay earth mud loam

clean pure spotless immaculate unspoiled
```

Mit dem Parameter *MATCH_SYNONYM* kann bei der Suchanfrage eine Synonymliste zur Ähnlichkeitsberechnung der einzelnen Wörter herangezogen werden, nachdem diese wie folgt im Datenbanksystem registriert wurde.

Syntax 12 Registrierung einer Synonymliste

```
EXECUTE PROCEDURE etx_CreateSynWlst
  ('my_synonymlist', '/local0/excal/syn_file', 'sbsp2')
```

Durch den Aufruf der Funktion *etx_CreateSynWlst()* wird die Synoymliste aus der Datei */local0/excal/syn_file* im Datenbanksystem unter dem Namen *my_synonymlist* im Datenbank-Speicherbereich *sbsp2* abgelegt.

3.2.1.3 CharacterSets

Ein CharacterSet definiert eine Abbildung, die angibt, wie ein einzelner Buchstabe oder ein einzelnes Zeichen des Dokuments in den Index aufzunehmen ist. Mit dem Parameter *CHAR_SET* können zur Indexierung die bereits vorhandenen CharacterSets für *ASCII* oder *ISO* verwendet werden.

Benutzerdefinierte CharacterSets werden wiederum als Textdatei angelegt. In einer 16x16-Matrix wird spezifiziert, wie einer der 256 möglichen Werte eines Zeichens in den Index übernommen wird. D.h. Ordinalwert x and Position y bedeutet, daß ein Zeichen mit Ordinalwert y als Zeichen mit Ordinalwert x in den Index aufgenommen wird. Das somit definierte CharacterSet wird mit den Funktionen *etx_CreateCharSet()* und *etx_DropCharSet()* in das System geladen oder wieder von dort entfernt.

Die folgende Syntax 13 zeigt nach [IFMXEX01] den möglichen Inhalt einer Datei zur Definition eines CharacterSets. Dort wird beschrieben, daß der Bindestrich (Wert 2D an Position 2D) im Index auf sich selbst abgebildet wird, ebenso die Ziffern 0 bis 9 (Werte 30 bis 39). Die Klein- und Großbuchstaben (Positionen 41 bis 5A und 61 bis 7A) werden jeweils auf die Großbuchstaben abgebildet (Werte 41 bis 5A).

Syntax 13 CharacterSet Datei

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 2D 00 00
30 31 32 33 34 35 36 37 38 39 00 00 00 00 00 00
00 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 00 00 00 00 00
00 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
50 51 52 53 54 55 56 57 58 59 5A 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Eine Datei diesen Typs kann nun unter einem beliebigen Namen als CharacterSet registriert werden. Dies zeigt nach [IFMXEX01] die folgende Syntax 14. Die Datei */local0/excal/my_new_char_set_file* wird unter dem Namen *my_new_charset* in die Datenbank geladen.

Syntax 14 Registrieren eines CharacterSets

```
EXECUTE PROCEDURE etx_CreateCharSet
  (‘my_new_charset’, ‘/local0/excal/my_new_char_set_file’)
```

3.2.2 Typen der Ähnlichkeitssuche

Das Excalibur Text Search DataBlade unterstützt verschiedene Typen der unscharfen Suche. Diese sind jedoch nicht als vollkommen verschiedene Verfahren zu verstehen, sondern bauen aufeinander auf, bzw. verfeinern das Ergebnis.

Das Ansprechen der Textsuchmaschine ist mit dem Operator *etx_contains()* realisiert, der als Parameter die Tabellenspalte, die die Textdokumente enthält, und einen Parameterstring erwartet.

Der Operator wird innerhalb der WHERE-Klausel einer SQL-Anfrage aufgerufen und kann dadurch mit weiteren Prädikaten verknüpft werden.

Wir betrachten nachfolgend das Verfahren der Mustersuche, der Wortmengensuche und der Nachbarschaftssuche.

3.2.2.1 Mustersuche

Die Mustersuche (pattern search) sucht innerhalb von Dokumenten nach Wortvorkommen, die einem spezifizierten Suchbegriff ähnlich sind. Dazu muß ein Index mit dem Parameter *WORD_SUPPORT=PATTERN* vorhanden sein. Zur Berechnung einer Ähnlichkeit wird dazu die *Substitution* und *Transposition* herangezogen.

Substitution. Unter einer Substitution verstehen wir nach [IFMXEX01] das Ersetzen einzelner Buchstaben innerhalb eines Wortes, um damit eine Übereinstimmung des Suchbegriffs mit einem Wort der Indexdaten zu erreichen. Beispiel für eine Substitution ist die Kontruktion des Wortes *Editer* aus dem Wort *Editor.*

Transposition. Unter einer Transposition verstehen wir nach [IFMXEX01] das Vertauschen einzelner Buchstaben innerhalb eines Wortes. Somit erreichen wir eine Übereinstimmung des Suchbegriffs mit Wörtern aus den Indexdaten, auch wenn im Suchbegriff Tippfehler durch Buchstabendreher vorhanden sind. Das falsch geschriebene Wort *Mulitmedia* statt *Multimedia* ist ein Beispiel für eine Transposition.

Die Verfahren der Substitution und Transposition zur Unterstützung der Suche können einzeln mit den Parametern *PATTERN_SUBS* und *PATTERN_TRANS* (oder beide durch *PATTERN_ALL*) bei der Suchanfrage aktiviert werden. Um einen Ähnlichkeitswert zwischen zwei Wörtern zu errechnen, wird berücksichtigt, wie oft Substitution und Transposition angewendet werden mußten, um den Suchbegriff auf ein Wort aus den Indexdaten abzubilden. Dieser so bestimmte Ähnlichkeitswert wird nach [IFMXEX01] als *Word Scoring* bezeichnet.

Nach Vorgabe werden bei einer Suchanfrage alle Dokumente zurückgeliefert, die beim *Word Scoring* einen Wert größer gleich 70 erreichen. Dieser Grenzwert kann jedoch bei der Anfrage auch explizit durch den Parameter *WORD_SCORE* vorgegeben werden.

Das folgende Beispiel zeigt eine Suchanfrage auf beschreibende Daten von Videofilmen. Es werden alle Dokumente gesucht, die die Wörter *multimedia*, *document* oder *editor* enthalten; dabei soll die Substitution, die Transposition und ein *Word Scoring* von größer gleich 85 verwendet werden.

Syntax 15 Suchanfrage mit Mustersuche

```
SELECT id, description
FROM videos
WHERE etx_contains (description, Row('multimedia document editor',
                                    'PATTERN_TRANS & PATTERN_SUBS & WORD_SCORE=85'))
```

3.2.2.2 Wortmengensuche

Die Wortmengensuche kann verwendet werden, wenn der Index auf der entsprechenden Textspalte der Tabelle mit dem Parameter *PHRASE_SUPPORT* und den Werten *MEDIUM* oder

MAXIMUM angelegt wurde. Dabei wird nun zusätzlich zu den Möglichkeiten der Substitution und Transposition die Reihenfolge der Wörter in der Suchanfrage berücksichtigt. Das Verhalten des Suchverfahrens kann mit dem Parameter *SEARCH_TYPE* und dessen Werten *PHRASE_EXACT* und *PHRASE_APPROX* gesteuert werden.

PHRASE_EXACT. Bei Anwendung der *PHRASE_EXACT* Option müssen alle Wörter der Suchanfrage in der angegebenen Reihenfolge in einem Dokument vorkommen, damit das Dokument in die Ergebnismenge übernommen wird. Zum Abgleichen der einzelnen Wörter kann nach wie vor die Substitution und Transposition verwendet werden.

PHRASE_APPROX. Die *PHRASE_APPROX* Option liefert auch Dokumente zurück, in denen die in der Anfrage spezifizierten Wörter nur teilweise oder in anderer Reihenfolge auftreten. Dabei verringert ein Abweichen von der vorgegebenen Reihenfolge oder das Fehlen eines Wortes den Ähnlichkeitswert.

Das folgende Beispiel nach [IFMXEX01] zeigt eine Suchanfrage auf Dokumente, die die Wörter *multimedia*, *document* und *editor* enthalten sollen. Die Wörter sollen in dieser Reihenfolge auftreten, Substitution und Transposition sollen verwendet werden.

Syntax 16 Suchanfrage mit Wortmengensuche

```
SELECT title
FROM reports
WHERE etx_contains (abstract, Row('multimedia document editor',
                                   SEARCH_TYPE=PHRASE_EXACT & PATTERN_ALL'))
```

3.2.2.3 Nachbarschaftssuche

Die in Abschnitt 3.2.2.2 beschriebene Wortmengensuche kann durch die Nachbarschaftssuche weiter verfeinert werden. Dazu steht der Parameter *SEARCH_TYPE* mit dem Wert *PROX_SEARCH(value)* zur Verfügung.

Die Nachbarschaftssuche bewirkt, daß alle Begriffe, die in der Suchanfrage spezifiziert sind, innerhalb von *value* Wörtern in einem Dokument auftreten müssen, damit sich dieses für das Suchergebnis qualifiziert. Ist der Wert von *value* kleiner als die Anzahl der Wörter in der Anfrage, so liefert der Aufruf des Operators *etx_contains()* eine Fehlermeldung.

Die folgende Anfrage nach [IFMXEX01] startet eine Suche nach Reporten, in deren Kurzfassung die Begriffe *multimedia* und *editor* innerhalb eines Bereichs von 5 Wörtern auftreten.

Syntax 17 Suchanfrage mit Nachbarschaftssuche

```
SELECT title
FROM reports
WHERE etx_contains (abstract, Row ('multimedia editor',
                                   'SEARCH_TYPE=PROX_SEARCH(5)'))
```

3.2.3 Anpassen der Suchergebnisse

Die gefundenen Dokumente, die die spezifizierte Suchanfrage voll oder teilweise mit einer entsprechenden Ähnlichkeit kleiner 100 erfüllen, werden zunächst ungeordnet und unabhängig von ihrem Ähnlichkeitswert ausgegeben.

Man kann jedoch die Dokumente nach ihrem Ähnlichkeitswert (dem sogenannten *Document Score*) sortieren lassen oder die Anzahl der zurückgelieferten Dokumente begrenzen. Es ist auch möglich, die Fundstellen der Suchbegriffe innerhalb der einzelnen Dokumente durch benutzerdefinierte Formatierungselemente hervorzuheben. Diese Möglichkeiten werden in den folgenden Abschnitten erläutert.

3.2.3.1 Document Score

Die Ähnlichkeit des Dokuments zur Suchanfrage wird durch den sogenannten *Document Score* ausgedrückt, ein Zahlenwert zwischen 0 und 100.

Der *Document Score* kann durch den optionalen dritten Parameter des etx_contains() Operators ermittelt werden. Dieser Parameter ist vom Typ *etx_ReturnType* und besitzt ein Feld namens *score*, das den *Document Score* enthält. Mittels einer lokalen Statementvariablen kann der *Document Score* abfragt werden und das Ergebnis danach sortiert werden.

Die folgende Syntax 18 zeigt nach [IFMXEX01] ein Beispiel, wie Dokumente nach dem *Document Score* sortiert ausgegeben werden können.

Syntax 18 Sortierte Ausgabe mit dem Document Score

```
SELECT rc.score, id, description
FROM videos
WHERE etx_contains (description, Row ('multimedia',
                                       'PATTERN_TRANS & PATTERN_SUBS'),
                    rc # etx_ReturnType)
ORDER BY 1
```

3.2.3.2 Einschränken der Ergebnismenge

Um die Anzahl der Dokumente in der Ergebismenge zu begrenzen, stehen mehrere Möglichkeiten zur Verfügung, die wir im folgenden kurz beschreiben.

MAX_MATCHES. Mit dem Parameter *MAX_MATCHES* wird auf einfache Weise die maximale Anzahl der Dokumente auf einen angegebenen Wert eingeschränkt. Unabhängig von der Qualität der Suchergebnisse (*Document Score*) wird nach Erreichen der angegebenen Zahl von Dokumenten die Suche abgebrochen.

Das folgende Beispiel nach [IFMXEX01] verdeutlicht die Verwendung des *MAX_MATCHES* Parameters in einer Suchanfrage, bei der nur 3 Dokumente zurückgeliefert werden sollen.

Syntax 19 Einschränkung der Ergebnismenge mit MAX_MATCHES

```
SELECT id, description
FROM videos
WHERE etx_contains (description, Row('multimedia document editor',
                                   'PATTERN_TRANS & PATTERN_SUBS & MAX_MATCHES=3'))
```

WORD_SCORE. Wie schon in Abschnitt 3.2.2.1 über die Mustersuche angesprochen, kann durch den Parameter WORD_SCORE beeinflußt werden, ob ein Wort als ähnlich gilt (Ähnlichkeitswert größer gleich dem definierten WORD_SCORE) oder nicht. Damit wird natürlich auch geregelt, ob ein Dokument, das nun mehrere Wörter einer Suchanfrage enthalten soll, in die Ergebnismenge aufgenommen wird oder nicht.

Document Score. Auch der in Abschnitt 3.2.3.1 eingeführte *Document Score* läßt sich für eine Einschränkung der Ergebnismenge heranziehen, indem eine Grenzwertbedingung in die WHERE-Klausel der Anfrage aufgenommen wird.

Die folgende Syntax 20 nach [IFMXEX01] verdeutlicht, wie nur Dokumente mit einem *Score* von über 85 in die Ergebnismenge übernommen werden.

Syntax 20 Einschränkung der Ergebnismenge mit Document Score

```
SELECT rc.score, id, description
FROM videos
WHERE etx_contains (description, Row ('multimedia document editor',
                                      'PATTERN_TRANS & PATTERN_SUBS')
                    rc # etx_ReturnType)
  AND rc.score > 85
```

3.2.3.3 Highlighting

Unter *Highlighting* versteht man das Hervorheben relevanter Textstellen innerhalb eines Dokuments. Das Excalibur Text Search DataBlade stellt hierzu die Funktionen *etx_GetHilite()* und *etx_ViewHilite()* zur Verfügung, mit denen eine Hervorhebung nach Benutzervorgaben möglich wird.

Die folgende Syntax demonstriert den Einsatz von *Highlighting*.

Syntax 21 Highlighting

```
SELECT etx_ViewHilite (etx_GetHilite (abstract, rc), '<b>', '</b>')
FROM reports
WHERE etx_contains (abstract, Row ('multimedia editor',
                                   'SEARCH_TYPE=PROX_SEARCH(5)'),
                    rc # etx_ReturnType)
```

Die Funktion *etx_GetHilite()* liefert als Ergebnis einen Datentyp mit zwei Feldern, in denen die Anfangspositionen und Längen der relevanten Textstellen vermerkt sind. Zusammen mit der

Funktion *etx_ViewHilite()* werden diese Textstellen durch die als Parameter übergebenen Formatierungselemente markiert.

3.2.4 Beispiel

Nach [IFMXEX01] verdeutlichen wir nun nochmal an einem Beispiel die Definition eines Indizes, der die Ähnlichkeitssuche auf Textdokumenten unterstützt und eine Suchanfrage mit Verwendung einer Synonymliste.

Zunächst muß eine Tabelle definiert werden, die die zu durchsuchenden Textdokumente und beschreibenden Daten verwaltet.

Syntax 22 Definition der Tabelle

```
CREATE TABLE
(
  doc_no    INTEGER,
  author    VARCHAR(60),
  title     CHAR(255),
  abstract  CLOB
)
```

Nachdem die in Abschnitt 3.2.1.1 und Abschnitt 3.2.1.3 eingeführte Stopwortliste und CharacterSet registriert worden sind, kann ein Index definiert werden.

Syntax 23 Definition des Index

```
CREATE INDEX reports_idx ON reports (abstract etx_clob_ops)
  USING etx (WORD_SUPPORT='PATTERN',
             STOPWORD_LIST='my_stopwordlist',
             PHRASE_SUPPORT='MAXIMUM',
             CHAR_SET='my_new_charset')
  IN sbsp1
```

Der hier definierte Index *reports_idx* unterstützt nun die Ähnlichkeitssuche auf Wörtern und die höchste Genauigkeit bei der Wortmengensuche. Es wird die benutzerdefinierte Stopwortliste *my_stopwordlist* und das benutzerdefinierte CharacterSet *my_new_charset* verwendet. Der Index wird im Datenbank-Speicherbereich *sbsp1* angelegt.

Nun kann eine Suchanfrage gestartet werden, die alle Dokumente zurückliefert, die die Worte *plug and play* in der angegebenen Reihenfolge enthalten sollen. Dazu soll die benutzerdefinierte Synonymliste *my_synonymlist* aus Abschnitt 3.2.1.2 verwendet werden.

Syntax 24 Suchanfrage mit benutzerdefinierter Synonymliste

```
SELECT title
FROM reports
WHERE etx_contains (abstract,
                    Row ('plug and play',
                       'SEARCH_TYPE=PHRASE_EXACT & MATCH_SYNONYM='my_synonymlist'))
```

3.3 Die Ansätze von Oracle

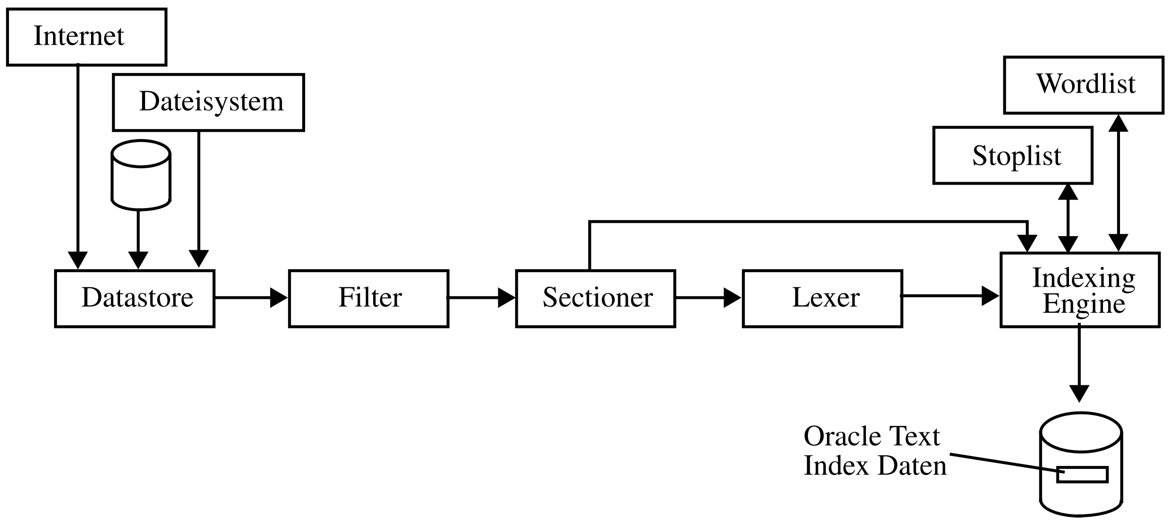
Auch Oracle bietet mit *Oracle Text* für das Datenbanksystem *Oracle9i* die Möglichkeit, auf Textdokumenten ähnlichkeitsbasierte Suchanfragen auszuführen ([OTFO01]). Dazu werden die Textdokumente wiederum in einzelnen Tabellenspalten verwaltet, um die Operatoren der Suche aufrufen zu können, muß ein spezieller Index auf der entsprechenden Spalte definiert sein.

In den folgenden Abschnitten betrachten wir die Komponenten, die ein Textdokument bei der Indexierung durchläuft, die Indextypen, die von Oracle zur Unterstützung der Suche zur Verfügung gestellt werden, und beenden schließlich die Diskussion von Oracle 9i Text mit einem Beispiel zur Indexdefinition und darauf aufbauender Suchanfrage.

3.3.1 Architektur zur Indexierung

Die folgende Abbildung 9 zeigt die Architektur der Oracle Text Komponenten, die ein Textdokument zur Indexierung durchläuft. Diese Komponenten werden nach [OTWP01] Objekte genannt und in den folgenden Abschnitten genauer erläutert.

Abbildung 9 Architektur



3.3.1.1 Das Datastore Objekt

Der zu indexierende Text kann aus verschiedenartigen Quellen stammen, die dem Datenbanksystem natürlich zugänglich sein müssen.

Datenbank. Der Text liegt direkt in Tabellen innerhalb des Datenbanksystems vor. Dazu werden nach [OTAD01] die Datentypen VARCHAR2, BLOB, CLOB, BFILE und XMLType unterstützt.

Dateisystem. Der zu indexierende Text kann als Datei in einem Dateisystem liegen. Dieses muß für den Datenbankserver zugänglich sein.

Internet. Textdokumente können im Internet zur Verfügung gestellt werden, und müssen zur Indexierung vom Datenbankserver über das HTTP- oder FTP-Protokoll erreichbar sein.

Benutzerdefiniert. Der zu indexierende Text ist der Rückgabewert einer benutzerdefinierten Funktion. Damit kann der Anwender aus beliebigen Daten individuell die zu indexierenden Daten extrahieren.

3.3.1.2 Das Filter Objekt

Das Filter Objekt dient zur Strukturierung des zu indexierenden Textes. Eingabe des Filter Objekts sind Textdokumente, die sich vom einfachen Rohtext über XML- oder PDF-Dateien bis hin zu Microsoft Word Dokumenten erstrecken.

Oracle Text stellt Filter für über 150 Dateiformate zur Verfügung, so daß nahzu jede Datei zur Indexierung geeignet strukturiert werden kann. Sollte dies nicht ausreichen, besteht zusätzlich die Möglichkeit, eigene Filter zu entwerfen oder Filter von Drittanbietern einzusetzen.

Der Einsatz von Filtern erlaubt in einem Dokument das Entfernen von Daten, die nicht zur Indexierung beitragen. Somit können beispielsweise Textformatierung, die zweifelsohne dateitypspezifisch sind, entfernt werden. Die Ausgabe eines Filters ist ein Textdokument, das im HTML-Format vorliegt.

3.3.1.3 Das Sectioner Objekt

Das Sectioner Objekt identifiziert Bereiche innerhalb von Dokumenten. Typischerweise sind dies die bereits vordefinierten Elemente von HTML-Dokumenten oder benutzerdefinierte Elemente in XML-Dokumenten.

Die Bearbeitung des zu indexierenden Textes mit dem Sectioner ermöglicht die spätere Suche nach Textvorkommen innerhalb der definierten Elemente.

3.3.1.4 Das Lexer Objekt

Der Lexer wandelt die Daten, die er vom Sectioner erhält, in die reine zu indexierende Textinformation um. Nach einem Beispiel aus [OTWP01] geht er dabei folgendermaßen vor.

Der Lexer erhält die folgende alphanumerische Zeichenkette als Eingabe.

Syntax 25 Eingabe für den Lexer

```
Aha! It's the 5:15 train, coming here now!
```

Der erste Schritt besteht darin, die Zeichenkette in einzelne kleingeschriebene Wörter zu zerlegen und dabei Sonderzeichen und Interpunktionssymbole zu entfernen. Dies ergibt dann die nach Syntax 26 folgende Wortliste.

Syntax 26 Lexer Wortliste

```
aha it s the 5 15 train coming here soon
```

Danach werden die Stopwörter entfernt, die nicht indexiert werden sollen. Welches Wort dabei als Stopwort gilt, wird wie auch bei den bereits vorgestellten Verfahren bei der *DB2* und dem *Informix Dynamic Server* aus einer Stopwortliste ermittelt. Danach liegen die folgenden Textdaten vor.

Syntax 27 Anwenden der Stopwortliste

```
aha * * * 5 15 train coming * now
```

Hierbei ist noch zu beachten, daß ein Stern ein entferntes Stopwort repräsentiert. Die Stopwörter werden zwar nicht indexiert, jedoch wird vermerkt, an welcher Position sich ein entferntes Stopwort befunden hat. Dies hat nach [OTWP01] den Vorteil, daß eine Suchanfrage nach *kicking the ball* den Text *kicking a ball*, aber nicht den Text *kicking ball* zurückliefern wird.

Der Lexer kann zusätzlich mit zahlreichen Optionen zur Wortverarbeitung betrieben werden, einige davon stellen wir nachfolgend mit ihren Bezeichnungen nach [OTWP01] kurz vor.

Join characters. Für Zeichen, wie zum Beispiel ein Apostroph, das zwei Wortteile miteinander verbindet, kann festgelegt werden, ob das ursprüngliche Wort unverändert oder ohne das verbindende Zeichen indexiert wird. Für *it's* wird somit eingestellt, ob es unverändert oder als *its* in den Index aufgenommen wird.

Base Letter Conversion. Akzentuierte Buchstaben werden auf ihre Grundform (Darstellung ohne Akzent) zurückgeführt und im Index abgelegt.

Alternate Spelling. In Sprachen, wie zum Beispiel im Deutschen, können Wörter in verschiedenen Schreibweisen dargestellt werden. So sollten *München* und *Muenchen* miteinander identifiziert werden. Ist die *Alternate Spelling* Option aktiviert, so würden beide Wörter als *Muenchen* im Index abgelegt werden.

Compound Word Processing. In Sprachen wie Deutsch oder Holländisch werden viele Wörter benutzt, die aus Teilwörtern zusammengesetzt sind. Beim *Compound Word Processing* werden solche Wörter wieder in ihre Einzelkomponenten zerlegt und separat indexiert.

3.3.1.5 Das Indexing Engine Objekt

Die Indexing Engine führt schließlich das Einfügen der durch die zuvor beschriebenen Objekte erzeugten Indexdaten durch, die in der gleichen Datenbank wie die Textdaten abgelegt werden. Dadurch wird eine hohe Leistung bei der Mischung von Ähnlichkeitsanfragen mit konventionellen SQL-Prädikaten erreicht, außerdem sind die Daten leichter zu verwalten (zum Beispiel bei der Datensicherung).

3.3.2 Indextypen

Oracle Text stellt drei Indextypen zur Verfügung, die unterschiedliche Einsatzbereiche abdecken: Den Standardindex, den Katalogindex und den Klassifikationsindex. Diese Indextypen werden in den folgenden Abschnitten 3.3.2.1 bis 3.3.2.3 kurz vorgestellt.

3.3.2.1 Standardindex

Der Standardindex unterstützt die ähnlichkeitsbasierte Suche auf großen Textdokumenten oder Webseiten, dabei werden (wie in Abschnitt 3.3.1.2 beschrieben) zahlreiche Dokumentformate unterstützt. Dieser Indextyp wird als *CONTEXT* bezeichnet.

Der Standardindex kann auf Tabellenspalten definiert werden, die vom Typ CLOB, BLOB, BFILE, VARCHAR2 oder CHAR sind. Die folgende Syntax 28 zeigt die Definition eines Standardindizes auf der Spalte *text* der Tabelle *docs*.

Syntax 28 Standardindex Definiton

```
CREATE INDEX myindex ON docs(text)
INDEXTYPE IS ctxsys.context
```

3.3.2.2 Katalogindex

Der Katalogindex eignet sich zur Indexierung kleiner Textfragmente, die durch ergänzende Information in weiteren Tabellenspalten ergänzt werden (zum Beispiel Artikelname, -beschreibung und -preis). Dieser Indextyp eignet sich für eBusiness Anwendungen. Dieser Typ wird als *CTXCAT* bezeichnet.

Der Katalogindex wird basierend auf Subindizes definiert, die jeweils verschiedene Anfrageklassen unterstützen sollen. Nach [OTAD01] verdeutlichen wir dies am Beispiel einer Auktionsanwendung. Syntax 29 zeigt die Definition einer Tabelle zur Verwaltung von Auktionsartikeln.

Syntax 29 Definition der Auktionstabelle

```
CREATE TABLE auction
(
  item_id      number,
  title        varchar2 (100),
  category_id  number,
  price        number,
  bid_close    date
)
```

Wir gehen davon aus, daß Applikationen, die auf diese Tabelle zugreifen, hauptsächlich Klauseln auf *title* in Verbindung mit *price* bzw. *price* und *bid_close* benutzen. Daher wollen wir einen

Katalogindex auf *title* anlegen, der einen Subindex *A* auf *price* und einen Subindex *B* auf *price* und *bid_close* verwendet.

Zunächst muß ein sogenanntes *index_set* erzeugt werden, das die Subindizes verwaltet. In Syntax 30 wird dazu ein *index_set* namens *auction_iset* definiert.

Syntax 30 Erzeugen des index_set

```
ctx_dll.create_index_set ('auction_iset')
```

Nun kann der Subindex *A* für *price* und der Subindex *B* für *price* und *bid_close* im index_set *auction_iset* angelegt werden. Die dazu durchzuführenden Funktionsaufrufe sind in Syntax 31 abgebildet.

Syntax 31 Definition der Subindizes

```
ctx_dll.add_index ('auction_iset', 'price');
ctx_dll.add_index ('auction_iset', 'price, bid_close');
```

Nun kann der Katalogindex auf dem oben eingerichteten index_set auction_iset definiert werden.

Syntax 32 Definition des Katalogindex

```
CREATE INDEX auction_title ON auction(title)
INDEXTYPE IS ctxcat
PARAMETERS ('index set auction_iset')
```

3.3.2.3 Klassifikationsindex

Der Klassifikationsindex dient Applikationen, die einzelne Textdokumente anhand deren Inhalt klassifizieren sollen. Der Indextyp wird als *CTXRULE* bezeichnet und indiziert eine Tabelle, die eine Menge von Anfrageklauseln verwaltet.

Wir wollen auf diese Funktionalität kurz nach einem Beispiel aus [OTAD01] eingehen, bei dem eingehende Nachrichtenmeldungen nach deren Themengebiet klassifiziert werden sollen. Zunächst legen wir in Syntax 33 eine Tabelle an und speichern darin die Nachrichtenkategorien mit entsprechenden Anfrageklauseln.

Syntax 33 Tabelle mit Anfrageklauseln

```
CREATE TABLE myqueries
(
  queryid    NUMBER PRIMARY KEY,
  category   VARCHAR2 (30),
  query      VARCHAR2 (2000)
);

INSERT INTO myqueries VALUES (1, ‘US Politics’, ‘democrat or republican’);
INSERT INTO myqueries VALUES (2, ‘Music’,       ‘ABOUT(music)’);
INSERT INTO myqueries VALUES (3, ‘Soccer’,      ‘ABOUT(soccer)’);
```

Im nächsten Schritt wird nun die Tabelle mit den Anfrageklauseln indexiert.

Syntax 34 Indexierung der Klauseltabelle

```
CREATE INDEX ON myqueries(query)
INDEXTYPE IS ctxrule
```

Nun können eingehende Nachrichten klassifiziert werden. Wir gehen davon aus, daß diese Nachrichten in der Tabelle *news* gespeichert werden.

Syntax 35 Tabelle news

```
CREATE TABLE news
(
  newsid    NUMBER,
  author    VARCHAR2(30),
  source    VARCHAR2(30),
  article   CLOB
)
```

Auf der Tabelle *news* wird ein Trigger definiert, der nach Eingang einer Nachricht deren Kategorie mittels der Tabelle *myqueries* ermittelt und daraufhin einen Eintrag in der Tabelle *news_route* mit der *newsid* und einer entsprechenden Kategorie einfügt. Der Rumpf des Triggers hat dann die in Syntax 36 angedeutete Gestalt.

Syntax 36 Klassifizieren der Nachrichten

```
BEGIN
  FOR c1 IN (SELECT category
             FROM   myqueries
             WHERE  MATCHES (query, :new.article) > 0)
  LOOP
    INSERT INTO news_route (newsid, category) VALUES (:new.newsid, c1.category);
  END LOOP;
END
```

Durch das Anlegen des Klassifikationsindex auf der Tabelle *myqueries* wird der Operator *MATCHES* verfügbar, mit dessen Hilfe nun die Kategorie einer eingehenden Nachricht bestimmt werden kann. Das Besondere an dem Index vom Typ *ctxrule* ist, daß er eine Tabelle mit Anfragen indexiert, die wiederum Indexoperatoren enthalten können. Die Tabelle *myqueries* nach Syntax 33 beinhaltet beispielsweise zweimal den Operator *ABOUT*, um die Klassifikationseigenschaften *music* oder *soccer* auszudrücken.

3.3.3 Beispiel

Als abschließendes Bespiel betrachten wir noch kurz die Definition eines Standardindex und eine anschließende Suchanfrage mit dem *contains()* Operator.

Gegeben sei dazu nach [OTWP01] eine Tabelle, die Produktdaten beinhaltet.

Syntax 37 Produktdatentabelle

```
CREATE TABLE product_information
(
  product_id           NUMBER(6) PRIMARY KEY,
  product_name         VARCHAR2(50),
  product_description  VARCHAR2(2000),
  category             NUMBER(2),
  product_status       VARCHAR2(20),
  list_price           NUMBER(8,2)
)
```

Um nun auf der Spalte *product_description* eine ähnlichkeitsbasierte Suchanfrage durchführen zu können, muß darauf ein Index angelegt werden. Wir definieren dazu in Syntax 38 einen Standardindex mit Namen *description_idx*.

Syntax 38 Indexdefinition

```
CREATE INDEX description_idx ON product_information(product_description)
INDEXTYPE IS ctxsys.context
```

Nun starten wir eine Suchanfrage nach den Produkten, in deren Beschreibung sich das Wort *monitor* möglichst nahe an den Wörtern *high resolution* befindet, die Ergebisdokumente sollen nach dem Ähnlichkeitswert sortiert ausgegeben werden (*ranking*).

Syntax 39 Suchanfrage

```
SELECT score(1), product_id, product_name
FROM product_information
WHERE contains (product_description, ‘monitor NEAR "high resolution"’, 1) > 0
ORDER BY score(1) DESC
```

Dabei ist *SCORE()* nach [OTR01] ein Operator, der in Zusammenhang mit dem *CONTAINS* Operator verwendet werden kann, um den berechneten Ähnlichkeitswert anzusprechen. Der letzte Parameter der *CONTAINS* Anweisung ordnet dem Operator eine sogenanntes *Label* zu (in Syntax 39 *Label 1*). Der von diesem *CONTAINS* Operator berechnete Ähnlichkeitswert kann dann mit *SCORE* über das definierte *Label* (im obigen Fall durch *SCORE(1)*) ausgegeben und die Dokumente danach sortiert werden.

3.4 Fazit

Der Funktionsumfang zur ähnlichkeitsbasierten Suche auf Textdokumenten ist bei allen drei vorgestellten Datenbankerweiterungen nahezu gleich. Um eine Suche durchführen zu können, muß ein Index auf einer entsprechenden Spalte definiert werden. Das Erweiterungsmodul für die Suche stellt einen oder mehrere Operatoren zur Verfügung, die in eine SQL-Anfrage integriert werden oder als eigenständige Funktion aufgerufen werden können.

Zum Anlegen eines Indizes bzw. der Auswertung einer Suchanfrage werden sprachenspezifische Stopwortlisten und Thesauri eingesetzt, die eine möglichst passende Ergebnismenge gewährleisten sollen.

In allen vorgestellten Lösungen ist jedoch nur eine Suche auf jeweils einer Tabellenspalte, die das Textdokument enthält, möglich. Eine Suchanfrage auf mehreren Attributen mit einer Aggregation der Teilähnlichkeiten und anschließendem *Ranking* ist nicht möglich (*Interklassen-Ähnlichkeit*).

Ebenfalls ist keine Unterstützung typisch objekt-relationaler Elemente wie zum Beispiel Tabellenhierarchien oder komplex strukturierter Attribute vorhanden. Ähnlichkeitsmaße basierend auf Zugehörigkeit von Instanzen zu Tabellentypen können nicht berechnet werden (*Intraklassen-Ähnlichkeit*).

Daher müssen zur Realisierung einer zufriedenstellenden Ähnlichkeitssuche in objekt-relationalen Datenbanksystemen weitere Mechanismen vorgesehen werden, bei denen die in diesem Kapitel stellvertretend vorgestellten Verfahren lediglich einen Teilaspekt abdecken, nämlich die Berechnung lokaler Ähnlichkeitsmaße auf Textattributen.

Kapitel 4 Anforderungen

Wir betrachten im folgenden nun einige Anforderungen an ein objekt-relationales Datenbanksystem, das zur Ähnlichkeitssuche eingesetzt wird. Dazu diskutieren wir zunächst kurz, welche Möglichkeiten ein objekt-relationales Datenbanksystem zur Datenmodellierung bietet. Davon ausgehend, beschreiben wir die Funktionalität, die zur Berechnung von Ähnlichkeitswerten vorhanden sein sollte.

4.1 Datenmodellierung in objekt-relationalen Datenbanksystemen

Alle gespeicherten Daten in objekt-relationalen Datenbanksystemen werden in dafür angelegten Tabellen verwaltet. Eine Tabelle besteht aus einer festen Anzahl von Attributen, die jeweils von einem Basisdatentyp, einem komplexen (konstruierten) Datentyp oder einem benutzerdefinierten Datentyp sind ([SQL99a], [SQL99b]).

Benutzerdefinierte Datentypen müssen wir nicht weiter berücksichtigen, da diese immer auf einem Basisdatentyp oder komplexen Datentyp definiert sind.

Ist ein Attribut der Tabelle von einem Basisdatentyp, so handelt es sich um einen boole'schen Wert, eine Zeichenketten (String), einen numerischen Wert, ein binäres Objekt oder einen Datums-/Zeitwert. Bis auf den boole'schen Wert besitzt jeder der Basisdatentypen noch diverse verfeinerte Untertypen, die Berechnung einer Ähnlichkeit unterscheidet sich aber nicht von der Ähnlichkeitsberechnung des ursprünglichen Basistyps.

Unter komplexen Datentypen von Attributen verstehen wir zum einen mengenorientiere Datentypen wie Mengen (*Collection*) und Listen (*Array*), sowie zum anderen zusammengesetzte Datentypen, die wir nach [SQL99a] im folgenden mit *RowType* bezeichnen. Ein RowType besteht aus einer festgelegten Anzahl von Attributen, jedes dieser Attribute kann wiederum Basisdatentyp, komplexer oder benutzerdefinierter Datentyp sein. Dadurch kann der Benutzer beliebig verschachtelte Strukturen erzeugen, die für ein Anwendungsproblem möglichst geeignet sind.

Tabellen können getypt sein, d. h. eine Tabelle kann basierend auf einem vorher definierten RowType angelegt werden.

Eine Tabelle kann zusätzlich zur Definition durch ihre Attribute noch als *Subtabelle* von einer übergeordneten *Supertabelle* abgeleitet werden. Sie erbt dann alle Attribute der Supertabelle, die zu den lokal definierten Attributen ergänzt werden. Damit können nun Tabellenhierarchien (ähnlich einem Klassenkonzept) erzeugt werden. Suchanfragen auf eine Tabelle können auch Elemente der Subtabellen zurückliefern.

Ausgehend von den hier angesprochenen Möglichkeiten der Datenmodellierung, stellen wir nun einige Anforderungen zusammen, die eine ähnlichkeitsbasierte Suche auf eben diesen Strukturen erfüllen sollte.

4.2 Lokale Ähnlichkeitsmaße

Zur Berechnung von Ähnlichkeitswerten zwischen einzelnen Attributwerten benötigt man eine Menge von lokalen Ähnlichkeitsmaßen, die auf den jeweiligen Definitionsbereichen der Attribute definiert sind (siehe Kapitel 2).

Damit wird es möglich, Ähnlichkeitswerte zwischen numerischen Attributen, Zeichenketten und Datums-/Zeittypen zu berechnen. Für symbolische, mehrwertige und taxonomisch geordnete Attribute sind Mechanismen vorzusehen, um die Ähnlichkeiten zwischen einzelnen Attributwerten im voraus spezifizieren zu können. Diese Informationen müssen während der Anfrageabarbeitung dem System verfügbar sein.

Man erhält somit die Möglichkeit, Anfragen auf Tabellen auszuführen, bei denen einzelne Attribute möglichst ähnliche Werte zu den in der Anfrage definierten Vergleichswerten aufweisen.

Für Ähnlichkeitsmaße auf Strukturen mit komplexen Datentypen und Graph oder prädikatenlogischen Repräsentationen wird man individuelle Lösungen implementieren müssen, da die Auswertung stark von der konkreten (und damit eventuell sehr unterschiedlichen) Realisierung der Repräsentation im Datenbanksystem abhängt. Dazu kommt noch die Komplexität der Algorithmen, die bestenfalls (bei geeigneter Anforderung an die gespeicherten Datensätze) in polynomialer Zeit abgearbeitet werden können (siehe Kapitel 2) und somit nur eine individuell entworfene, approximierte Lösung zulassen.

4.3 Globale Ähnlichkeitsmaße

Da Tabellen mehrere Attribute enthalten, muß aufbauend auf den lokalen Ähnlichkeitsmaßen für einzelne Attribute deren Aggregation zu einem globalen Ähnlichkeitsmaß möglich sein. Dadurch können wir zwischen zwei Instanzen einen Ähnlichkeitswert berechnen, der durch den Einfluß mehrere Attribute gebildet wird.

Für jedes Attribut wird dabei eine lokale Ähnlichkeit berechnet. Aus den lokalen Ähnlichkeiten wird mittels einer Aggragtionsfunktion die globale Ähnlichkeit berechnet (siehe Abschnitt 2.2.9).

Im Zusammenhang mit Vererbungsbeziehungen zwischen den Tabellen erhalten wir somit die Möglichkeit, Ähnlichkeitswerte zwischen Objekten verschiedenen Typs zu berechnen. Dazu werden zur Ermittlung der globalen Ähnlichkeit nur die lokalen Ähnlichkeiten herangezogen, die für beide Objekttypen berechnet werden können. Man erhält auf diesem Weg die Möglichkeit, eine *Intra-Klassen Ähnlichkeit* (Abschnitt 2.2.9) zu berechnen.

Eine Anfrage kann dadurch auch Ergebnistupel liefern, deren Typ sich von der Vergleichsinstanz unterscheidet. Dazu müssen Vergleichsinstanz und Ergebnistupel einen gemeinsamen Vorgängerknoten in der Tabellenhierarchie haben oder aber eine Abbildung zwischen den Attributen der Tabellen existieren.

4.4 Berücksichtigung der Objekthierarchie

Um bei der Berechnung des globalen Ähnlichkeitsmaßes die Hierarchie der Tabellenstruktur zu berücksichtigen, ist es erforderlich, für jede Tabelle innerhalb ihrer Hierarchie einen maximalen Ähnlichkeitsfaktor zu verwalten.

Ähnlich den Taxonomien (Abschnitt 2.2.6) kann damit zusätzlich zur Ähnlichkeit der aggregierten Attributwerte noch ein grundsätzlicher Ähnlichkeitsfaktor aufgrund des Instanzentyps ermittelt werden, der in das Gesamtähnlichkeitsmaß einfließt. Der für jede Tabelle definierte Ähnlichkeitsfaktor gibt die Maximalähnlichkeit zweier Instanzen aus verschiedenen Ableitungszweigen dieser Tabelle an. Wir erhalten damit die Möglichkeit, eine *Inter-Klassen Ähnlichkeit* (Abschnitt 2.2.9) zu ermitteln.

Zusammenfassend können wir damit durch eine Anfrage alle Tupel innerhalb einer Tabellenhierarchie bestimmen, die eine definierte Mindestähnlichkeit aufgrund vorgegebener Attributwerte (Aggregation der lokalen Ähnlichkeiten zur globalen Intra-Klassen Ähnlichkeit) und der Klassenzugehörigkeit (Berechnung der Inter-Klassen Ähnlichkeit durch den tabellenspezifischen Ähnlichkeitsfaktor) aufweisen.

4.5 Parametrisierung

Je nach gegebener Anwendungsdomäne kann es sinnvoll sein, nicht für jeden Benutzer die gleichen Funktionen zur Berechnung von Ähnlichkeiten zu verwenden. Zudem kann es für den einzelnen Anwender bei seiner Recherche in der Datenbank hilfreich sein, durch gezielte Anpassung einzelner Funktionen das Suchergebnis zu verändern und seinen Bedürfnissen anzupassen.

Daher ist eine Parametrisierung des gesamten Prozesses der Ähnlichkeitssuche wünschenswert. Sowohl die verwendeten Algorithmen und deren Parameter der lokalen Ähnlichkeitsmaße, die Methoden der globalen Ähnlichkeitsmaße als auch die individuellen Ähnlichkeitsfaktoren der Tabellen innerhalb einer Hierarchie sollten vom Benutzer an seine Anforderungen angepaßt werden können.

Die Möglichkeit der Parametrisierung des Suchprozesses beeinflußt entscheidend die Qualität der Suchergebnisse einer ähnlichkeitsbasierten Suche und sollte daher zur Verfügung gestellt werden.

Kapitel 5 Realisierungsaspekte

In diesem Kapitel betrachten wir einige Realisierungsaspekte einer ähnlichkeitsbasierten Suche in objekt-relationalen Datenbanksystemen. Dabei konzentrieren wir uns auf die verschiedenen Möglichkeiten, die einem Benutzer zur Verfügung gestellt werden, um das Suchverfahren an seine persönlichen Anforderungen anzupassen.

Wir unterscheiden zwischen starker Parametrisierung, schwacher Parametrisierung und einer Ähnlichkeitssuche ohne modifizierbare Parameter für den Anwender und geben Verfahren an, mit denen ein Datenbanksystem Suchanfragen schneller verarbeiten kann.

5.1 Starke Parametrisierung

Unter starker Parametrisierung verstehen wir, daß jeder Benutzer das komplette Suchverfahren (so wie dies in Kapitel 4 gefordert wurde) vollständig an seine individuellen Bedürfnisse anpassen kann.

5.1.1 Eigenschaften

Das Datenbanksystem stellt für jeden Datentyp eine Auswahl von Ähnlichkeitsmaßen (siehe dazu Kapitel 2) zur Verfügung. Der Benutzer kann für jedes Attribut einer Tabelle (entsprechend dessen Datentyp) ein Ähnlichkeitsmaß wählen, mit dem der Ähnlichkeitswert zu anderen Instanzen oder einem Suchwert berechnet wird. Ist das Ähnlichkeitsmaß parametrisiert, so hat der Anwender die Möglichkeit, jeden Parameter der Ähnlichkeitsfunktion zu modifizieren; das System muß diese Einstellungen verwalten.

Um die Gesamtähnlichkeit eines Datensatzes zu einer Vergleichsinstanz mit mehreren Attributen zu berechnen, werden die einzelnen Attribute mit einem spezifischen Gewicht zu einem globalen Ähnlichkeitswert aggregiert. Dazu muß der Benutzer jedem Attribut ein Gewicht zuweisen können, das dessen Relevanz bei der Berechnung der Gesamtähnlichkeit ausdrückt. Die gewichteten Attribute werden schließlich mit einer Aggregationsfunktion zu einer globalen Ähnlichkeit zusammengefaßt. Auch hier sollte das Datenbanksystem dem Anwender mehrere Aggregationsfunktionen zur Auswahl anbieten. Da diese Funktionen wiederum parametrisierbar sind, müssen auch hier die Einstellungen des Benutzers verwaltet werden.

Objekt-relationale Datenbanksysteme bieten uns die Möglichkeit, Tabellen in Hierarchien unter Verwendung des Vererbungskonzepts zu organisieren. Bei Anwendung dieses Konzepts ist zu beachten, daß auch die oben beschriebenen Parameter und Benutzereinstellungen der Vererbung unterliegen und innerhalb einer Hierarchie propagiert werden müssen. Zusätzlich entsteht durch

Verwendung einer Tabellenhierarchie die Möglichkeit, zwischen Instanzen verschiedener Tabellen eine *Intra-Klassen Ähnlichkeit* zu berechnen (siehe Abschnitt 2.3). Dazu muß der Benutzer für jede Tabelle einen Ähnlichkeitsfaktor spezifizieren können, der die maximale Ähnlichkeit von Elementen verschiedener Zweige der Vererbungshierarchie angibt. Dieser Faktor fließt zur Berechnung in die Aggregationsfunktion ein.

5.1.2 Nachteile

Bei der Realisierung der starken Parametrisierung ergeben sich beim Leistungsverhalten und damit bei der Antwortzeit einer Suchanfrage große Nachteile.

Durch die große Anzahl der Parameter, die jederzeit vom Benutzer geändert werden können, ist es nicht möglich, geeignete Indexstrukturen zu verwalten, die schnell zu einem Suchergebis führen. Zum einen müßten getrennte Indexstrukturen für jeden Benutzer verwaltet werden, was enorme Kosten bei der Datenaktualisierung nach sich zieht, da auch die Indexdaten angepaßt werden müssen. Zum anderen müßten die Indizes nach jeder Änderung von Suchparametern neu aufgebaut werden, was zu einem nicht zumutbaren Systemverhalten gegenüber dem Benutzer führt.

Bei einer Suchanfrage müssen zunächst die aktuell vom Benutzer eingestellten Parameter bestimmt werden, mit denen danach die Ähnlichkeitsfunktionen aufgerufen werden. Wenn sich die Anfrage über mehrere Tabellen erstreckt, so muß die Parameterbestimmung für jede Tabelle getrennt durchgeführt werden (bei objekt-relationalen Datenstrukturen ist eventuell die Vererbung von Parametern zu berücksichtigen).

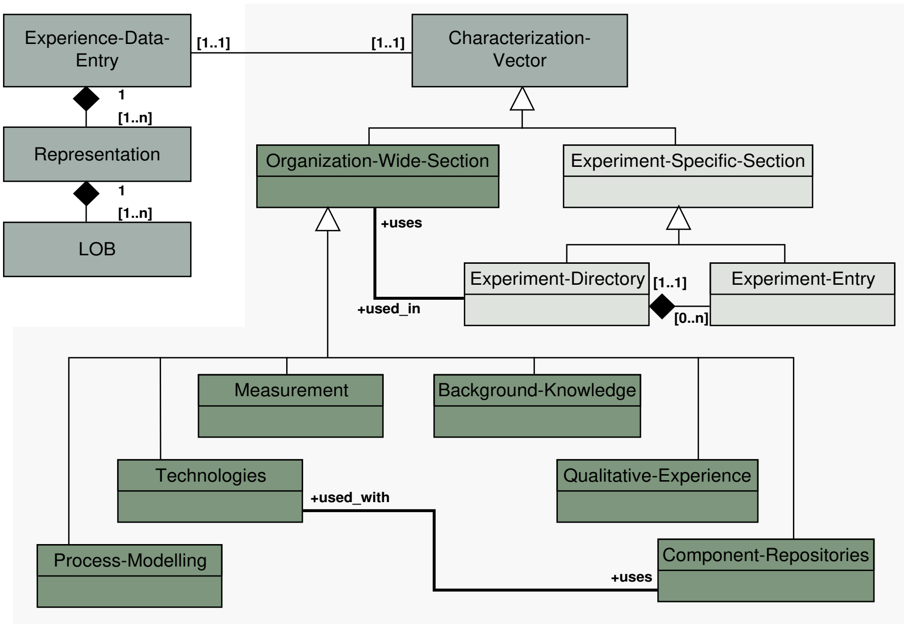
Da aus den oben genannten Gründen keine Indexdaten vorliegen, müssen die Ähnlichkeitsfunktionen nun für jede Datensatzinstanz einzeln aufgerufen werden, um einen Ähnlichkeitswert zu ermitteln. Dies ist besonders bei Ähnlichkeitsmaßen für Taxonomien sehr teuer, da ein Aufruf der Ähnlichkeitsfunktion zahlreiche Datenbankzugriffe zur Bestimmung der Ähnlichkeit zweier Taxonomiebegriffe zur Folge haben kann.

5.1.3 Beispiel SFB-501-Reuse-Repository

Das *Reuse Repository* des Sonderforschungsbereichs 501 ([RR00], [RR02]) bietet ein Beispiel für den Einsatz einer ähnlichkeitsbasierten Suche mit starker Parametrisierung. Abbildung 10 zeigt einen für unsere Betrachtungen relevanten Ausschnitt des Datenbankschemas.

Die Datensätze (sogenannte *Erfahrungselemente*) können in mehreren Repräsentationen mit jeweils mehreren binären Objekten gespeichert werden. Zu jedem Erfahrungselement existieren beschreibende Daten, die in genau einem *Charakterisierungsvektor* abgelegt sind. Diese Daten werden in einer hierarchischen Tabellenstruktur mit der Wurzel *Characterization-Vector* verwaltet. Auf dieser Tabellenhierarchie findet die Ähnlichkeitssuche statt.

Abbildung 10 SFB-501-Reuse-Repository



Für jedes Attribut eines Charakterisierungsvektors ist ein lokales Ähnlichkeitsmaß definiert. Die lokalen Ähnlichkeitsmaße s_i mit den Einzelgewichten w_i werden mit der gewichteten Durchschnittsaggregation Φ (siehe Abschnitt 2.2.9) zu einer globalen Gesamtähnlichkeit nach Abbildung 11 verrechnet.

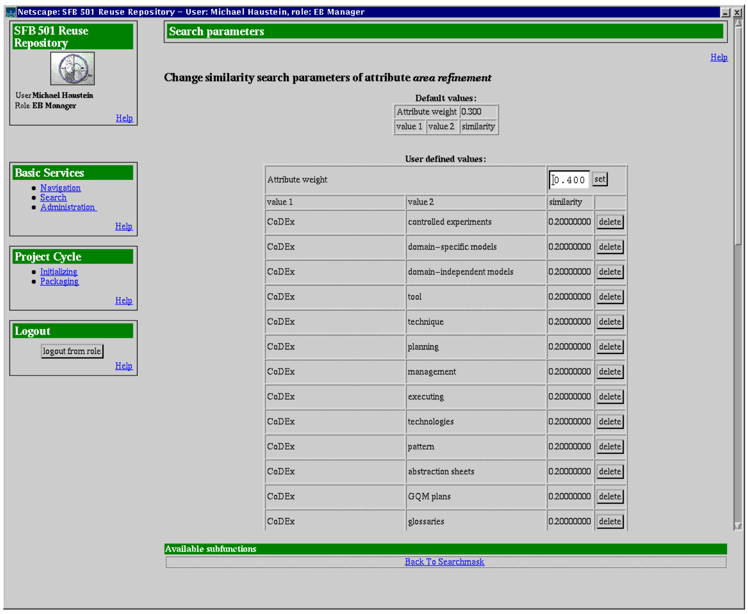
Abbildung 11 Berechnung der gewichteten Durchschnittsaggregation

$$\Phi(s_1, \ldots, s_n) = \sum_{i=1}^{n} (w_i \cdot s_i)$$

Jeder Benutzer kann jedes lokale Ähnlichkeitsmaß für jedes Attribut einer Tabelle parametrisieren. Dazu werden über das Web-Interface des Reuse Repository HTML-Seiten angeboten, über die die Parameter der Ähnlichkeitsfunktionen für die Datentypen *Integer*, *Date*, *String*, *ein-* und *mehrwertige Taxonomien* verwaltet werden können. Abbildung 12 zeigt die Administration der Parameter einer Ähnlichkeitsfunktion für Taxonomien.

Der Benutzer kann für jedes mögliche Wertepaar des Taxonomieattributs einen Ähnlichkeitswert definieren. Wird für zwei Werte keine Ähnlichkeit vom Benutzer angegeben, so wird die vom Reuse Repository Administrator definierte Standardähnlichkeit zur Berechnung verwendet. Zusätzlich kann ein Gewicht spezifiziert werden, mit dem das lokale Ähnlichkeitsmaß in die globale Ähnlichkeit eingeht.

Abbildung 12 Parameterverwaltung für Ähnlichkeitsmaße



Der Leistungsengpaß entsteht nun, da bei der Berechnung der lokalen Ähnlichkeit für zwei Attributwerte oft zwei Datenbankzugriffe erforderlich sind, um den Ähnlichkeitswert zu ermitteln. Zunächst wird nach einem vom Benutzer angegebenen Ähnlichkeitswert gesucht; schlägt diese Suche fehl, so muß die Ähnlichkeitsvorgabe des Administrators ermittelt werden, was einen weiteren Zugriff erfordert. Da die wenigsten Benutzer Werte für alle Attributpaare definieren, bzw. viele Attributwerte keine Ähnlichkeiten aufweisen (und somit auch keine Ähnlichkeitswerte definiert werden), sind in den meisten Fällen zwei Zugriffe nötig, um den lokalen Ähnlichkeitswert zu bestimmen.

5.1.4 Unterstützung der Ähnlichkeitsberechnung für Taxonomien

Um die Berechnung des Ähnlichkeitswertes für taxonomische Attribute etwas zu beschleunigen, wollen wir den Ähnlichkeitswert mit durchschnittlich einem Datenbankzugriff ermitteln.

Dazu legen wir eine Puffertabelle an, in die die Ähnlichkeitsfunktion den für zwei Attributwerte ermittelten Ähnlichkeitswert eingeträgt. Beim Aufruf der Ähnlichkeitsfunktion wird nun zunächst überprüft, ob in der Puffertabelle bereits ein Ähnlichkeitswert vorhanden ist. Ist dies der Fall, so wird dieser Wert zurückgeliefert und die Berechnung konnte mit einem Datenbankzugiff durchgeführt werden. Ist noch kein Ähnlichkeitswert in der Puffertabelle hinterlegt worden, so wird die Berechnung wie bisher durchgeführt und danach das Ergebnis in die Puffertabelle eingetragen. Das bedeutet, zuerst wird eine Suche nach benutzerdefinierten Parametern durchgeführt; endet diese erfolglos, so werden die Standardparameter bestimmt. Diese aufwendigere Abarbeitung fällt jedoch kaum ins Gewicht, da sie nur einmal für je zwei Attributwerte durchgeführt werden muß und (entsprechend viele Datensätze vorausgesetzt) durchschnittlich die Berechnung der Ähnlichkeit mit einem Zugriff erlaubt.

Der Inhalt der Puffertabelle muß durch zusätzliche Maßnahmen konsistent gehalten werden. Da der Benutzer vor jeder Suchanfrage die Parameter des Ähnlichkeitsmaßes verändern kann, das heißt, die Ähnlichkeit zweier Attributwerte neu definieren kann, muß der Inhalt der Puffertabelle spätestens bei jeder Parametermodifikation geleert werden. Alternativ könnte die Leerung der Puffertabelle natürlich auch direkt nach jeder Suchanfrage erfolgen.

Das Reuse Repository wurde um die oben beschriebene Funktionalität der Puffertabelle erweitert und darauf einige Leistungsmessungen durchgeführt.

Definiert ein Benutzer für alle möglichen Wertepaare eines Taxonomieattributs eigene Ähnlichkeitswerte, so kann eine Anfrage mit Einsatz einer Puffertabelle in etwa 87 Prozent der Anfragezeit ohne Puffertabelle durchgeführt werden. Zwar ist sowohl mit als auch ohne Puffertabelle zur Ähnlichkeitsbestimmung zweier Werte nur ein Datenbankzugriff erforderlich, ohne Puffertabelle ist aber ein Verbund über zwei Tabellen nötig, während die Puffertabelle die Berechnung mit direktem Zugriff auf nur einer Tabelle erlaubt.

Bei einer etwas typischeren Situation, in der der Benutzer nur für ein Drittel aller Wertepaare eigene Ähnlichkeitswerte definiert hat (weil er die Standardvorgaben akzeptiert oder die Wertepaare keinerlei Ähnlichkeit aufweisen), beträgt die Ausführungszeit mit einer Puffertabelle nur noch etwa 74 Prozent der ursprünglichen Zeit.

Die genaue Durchführung und Ergebnisse der Leistungsmessungen beim Einsatz einer Puffertabelle im Reuse Repository werden im Anhang A erläutert.

5.2 Schwache Parametrisierung

Bei der schwachen Parametrisierung schränken wir die Möglichkeiten des Benutzers, sämtliche Parameter des Suchvorgangs anzupassen, stark ein. Der Benutzer kann keine lokalen Ähnlichkeitsmaße mehr auswählen und auch die Parameter der datentyp-spezifischen Ähnlichkeitsmaße nicht mehr beeinflussen. Lediglich die Modifikation der Gewichte der einzelnen Attribute und die Wahl und Parametrisierung der Aggregationsfunktion zur Berechnung der Gesamtähnlichkeit kann vom Benutzer durchgeführt werden.

Im folgenden Abschnitten führen wir die inversen Indizes ein und diskutieren danach in den vier Abschnitten 5.2.2 bis 5.2.5 inverse Indizes für Taxonomien, Ganzzahl- und Fließkommadaten und Zeichenketten. Dabei erläutern wir die grundlegenden Ideen zur Konstruktion der Indexstrukturen und betrachten darauf aufbauend die Leistungsvorteile eines Indexzugriffs. In Abschnitt 5.2.6 gehen wir kurz auf das *Function-value-indexing* ein und beschreiben, wie diese Methode der Beschleunigung von Funktionsaufrufen für Ähnlichkeitsmaße eingesetzt werden kann. Abschließend betrachten wir die Aggregation lokaler Ähnlichkeiten und die Auswirkung von NULL-Werten auf die zu indexierenden Attribute.

5.2.1 Inverse Indizes

Die Vorgabe lokaler Ähnlichkeitsmaße mit festgelegten Parametern erlaubt uns die Unterstützung der Suchanfrage durch inverse Indizes mit teilweise ernormen Leistungssteigerungen. Inverse Indizes sind Indexstrukturen, die uns ausgehend von den Bedingungen einer Suchanfrage direkt die Primärschlüssel und Ähnlichkeitswerte qualifizierter Datensätze liefern. Es ist also nicht nötig, während einer Suchanfrage alle Tupel einer Datentabelle nach deren Qualifikation zu untersuchen, da durch das Vorhandensein der Indexdaten bereits zuvor berechnete Informationen zur Ähnlichkeitsermittlung vorliegen. Dabei existieren für jedes indexierte Attribut einer Tabelle separate Indexdaten.

Die Vorteile des schnellen Zugriffs mittels inverser Indizes können auch beim Einsatz von Tabellenhierarchien in objekt-relationalen Datenbanksystemen genutzt werden. Für ein Attribut, das über mehrere Tabellen weitervererbt wird, reicht es aus, eine einzige Indexstruktur zu verwalten. Selbst wenn in jeder Tabelle der Hierachie ein anderes lokales Ähnlichkeitsmaß für dieses Attribut definiert ist, können die Ähnlichkeitsinformationen in jeder der Tabellen individuell berechnet und danach in der gemeinsamen Indexstruktur abgelegt werden. Eine Suchanfrage nach einem Suchwert und einer Mindestähnlichkeit liefert dann die Primärschlüssel der qualifizierten Datensätze und eventuell auch die Angabe der Tabelle, in der der Datensatz abgelegt ist.

5.2.2 Inverser Index für Taxonomien

Taxonomien repräsentieren eine Beziehungsordnung von Begriffen durch eine Baumstruktur (siehe dazu Abschnitt 2.2.6). Die Ähnlichkeit zweier Begriffe kann aus dieser Struktur abgeleitet werden. Da es beim Aufruf einer Ähnlichkeitsfunktion für zwei als Parameter übergebene taxonomische Attribute nicht sinnvoll wäre, die Ähnlichkeit jedesmal aus der Baumstruktur neu zu bestimmen, bietet es sich auch ohne Einsatz eines Indizes an, alle möglichen Ähnlichkeitswerte zwischen Taxonomiebegriffen in einer Tabelle zu speichern.

Zur Verwaltung von Taxonomien und Ähnlichkeitswerten werden daher zwei Tabellen benötigt (siehe Syntax 40). Die Tabelle *taxtypes* speichert zu einer eindeutigen ID die Bezeichnung des Taxonomiebegriffs, die Tabelle *taxsims* verwaltet zu je zwei referenzierten Begriffen aus der Tabelle *taxtypes* eine Ähnlichkeit.

Syntax 40 Verwaltung von Taxonomien

```
CREATE TABLE taxtypes
(
  id    int PRIMARY KEY,
  name  char(10)
);

CREATE TABLE taxsims
(
  id1    int   REFERENCES taxtypes(id),
  id2    int   REFERENCES taxtypes(id),
  sim    decimal(5,2)
);
```

Die Ähnlichkeitsfunktion für Taxonomien bekommt nun als Parameter zwei Referenzwerte auf die Tabelle *taxtypes* übergeben und ermittelt durch eine einfache Anfrage auf der Tabelle *taxsims* die Ähnlichkeit beider Werte. Ebenso kann statt dem Aufruf einer Ähnlichkeitsfunktion in der SQL Anfrage der Verbund einer Datentabelle mit der Tabelle *taxsims* einen Ähnlichkeitswert liefern. Die folgende Syntax 41 zeigt eine Suchanfrage mit einem Verbund nach dem Taxonomiewert *3* und einer Mindestähnlichkeit von *0,9*.

Syntax 41 Suche mit Verbund ohne Indexunterstützung

```
SELECT d.tupleid, s.sim
FROM   taxdata d, taxsims s
WHERE  d.tupletype=s.id1
  AND  s.id2 = 3
  AND  s.sim > 0.9
ORDER BY s.sim DESC
```

Die Idee für einen Index auf taxonomischen Daten besteht nun darin, Ähnlichkeitswerte vorzuberechnen. Man speichert in einer Tabelle *taxidx*, welche Ähnlichkeit das Taxonomieattribut des Datensatzes einer Datentabelle zu einem gegebenen Taxonomiewert hat.

Syntax 42 Indextabelle taxidx

```
CREATE TABLE taxidx
(
  taxtype    int  REFERENCES taxtypes(id),
  tupleid    int  REFERENCES taxdata(tupleid),
  sim        decimal(5,2)
);
```

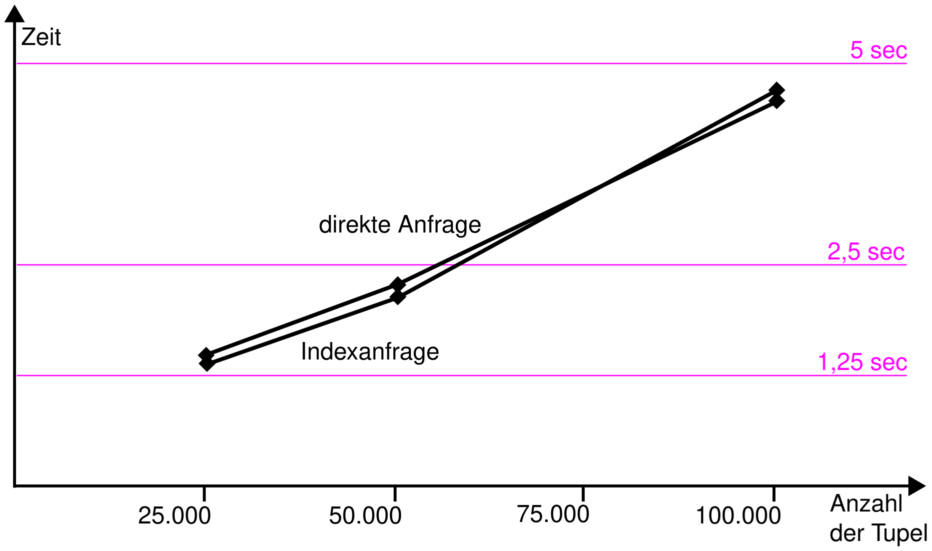
Somit muß die Suchanfrage nicht mehr über eine Ähnlichkeitsfunktion oder einen Verbund erfolgen, sondern kann allein auf der Indextabelle *taxidx* durchgeführt werden. Es wird direkt angefragt, welche Datensätze zu einem gegebenen Suchwert eine gegebene Mindestähnlichkeit aufweisen.

Syntax 43 Suche mit den Indexdaten

```
SELECT tupleid, sim
FROM   taxidx
WHERE  taxtype=3
  AND  sim > 0.9
ORDER BY sim DESC
```

Die Erstellung von Indexdaten mit vorberechneten Ähnlichkeitswerten bringt jedoch keine Leistungsvorteile gegenüber der Verbundanfrage mit einer Datentabelle. Die folgende Abbildung 13 zeigt die Ergebnisse einer Leistungsmessung, die für taxonomische Daten durchgeführt wurde.

Abbildung 13 Leistungsmessung mit taxonomischen Daten



Die Messung wurde mit 25.000, 50.000 und 100.000 Datensätzen durchgeführt, die Anfragezeiten mit und ohne Index weichen kaum voneinander ab. Die genaue Durchführung und Ergebnisse der Leistungsmessung für taxonomische Daten sind in Anhang B im Abschnitt B.1 zusammengetragen.

Die geringe Abweichung der Ausführungszeiten läßt sich zum einen darauf zurückführen, daß die Indextabelle sehr viele vorberechnete Datensätze beinhaltet, da zu jedem Taxonomiebegriff alle ähnlichen Datensätze in die Indextabelle eingetragen werden. Zum anderen muß der Ähnlichkeitswert nicht durch eine Berechnung bestimmt werden, sondern kann durch einen einfachen Verbund mit der Ähnlichkeitstabelle ermittelt werden. Dazu kann das Datenbanksystem alle seine Möglichkeiten der Anfrageoptimierung einsetzen.

Für taxonomische Daten, deren Ähnlichkeitswerte direkt aus einer Tabelle bestimmt werden können, ist der Einsatz eines inversen Indizes somit also nicht notwendig.

5.2.3 Inverser Index für ganzzahlige Daten

Für ganzzahlige Datentypen muß eine Ähnlichkeitsfunktion implementiert werden, die die Ähnlichkeit zwischen einem Attributwert und einem Suchwert berechnet. Eine Auswahl der dazu in Frage kommenden Funktionen wurde in Abschnitt 2.2.4 diskutiert. Die Ähnlichkeitsfunktion bekommt als Parameter zwei Zahlenwerte übergeben und berechnet damit einen Ähnlichkeitswert, der innerhalb einer SQL Anfrage zur Qualifikation eines Datensatzes herangezogen werden kann. Die folgende Syntax 44 zeigt eine solche Anfrage.

Syntax 44 Suchanfrage mit Ähnlichkeitsfunktion

```
SELECT id,data,simInt(data,2000) AS sim
FROM   intdata
WHERE  simInt(data,2000) > 0.7
ORDER BY sim DESC
```

Die Suchanfrage liefert alle Datensätze der Tabelle *intdata*, deren ganzzahliges Attribut *data* zum Suchwert *2000* eine Ähnlichkeit größer *0,7* aufweist. Bei der Verarbeitung dieser Anfrage muß nun für jeden Datensatz der Tabelle *intdata* mit einem Funktionsaufruf der Ähnlichkeitsfunktion *simInt()* ein Ähnlichkeitswert zum angegebenen Suchwert *2000* berechnet werden.

Das *Function-value-indexing* ([IFMXSQLa01]) erlaubt bei gewissen Einschränkungen eine erhebliche Leistungssteigerung, wie in Abschnitt 5.2.6 noch erläutert wird. Eine allgemeine Leistungsverbesserung der Abarbeitung einer Anfrage mit beliebigem Suchwert ist durch vorgegebene Mechanismen des Datenbanksystems allerdings nicht möglich, da der Funktionswert von *simInt()* für jeden Datensatz erst berechnet werden muß, und der Suchwert bei jeder Anfrage variieren kann.

Zur Leistungsunterstützung der ähnlichkeitsbasierten Suche auf ganzzahligen Daten wird ein inverser Index aufgebaut, dessen Inhalte von der zu benutzenden Ähnlichkeitsfunktion berechnet werden. Um die korrekte Erzeugung der Indexdaten zu gewährleisten, muß die Ähnlichkeitsfunktion zwei Bedingungen erfüllen.

Sei dazu c der Wert des ganzzahligen Attributs eines Datensatzes aus der Datentabelle und s eine zuvor definierte Mindestähnlichkeit. Dann muß die Ähnlichkeitsfunktion im Bereich $]-\infty, c]$ monoton steigend und im Bereich $[c, +\infty[$ monoton fallend sein. Zudem müssen zwei Grenzwerte a und b existieren, derart daß $a < c$ und $b > c$ und alle Ähnlichkeitswerte von d und c bzw. e und c für ein beliebiges $d \leq a$ und ein beliebiges $e \geq b$ kleiner sind als s.

Für Ähnlichkeitsfunktionen, die die soeben beschriebenen Voraussetzungen erfüllen, ist es nun möglich, Ähnlichkeitswerte im voraus zu berechnen und diese in einer Indexdatentabelle zu speichern. Die Indexdatentabelle *intidx* speichert zu einem Suchwert x alle Referenzen auf Datensätze, deren indexiertes Attribut zum Suchwert die spezifizierte Mindestähnlichkeit besitzt.

Syntax 45 Indextabelle intidx

```
CREATE TABLE intidx
(
  value   int,
  tupleid int  REFERENCES intdata(id),
  sim     decimal(5,2)
);
```

Um die Indexdaten zu erzeugen, wird der Attributwert eines Datensatzes in eine temporäre Variable kopiert und solange inkrementiert, bis die berechnete Ähnlichkeit zwischen dem Variablenwert und dem ursprünglichen Attributwert unter die angegebene Mindestähnlichkeit fällt. Alle während dieser schrittweisen Inkrementierung berechneten Ähnlichkeitswerte werden zusammen mit der Referenz zum Datensatz in der Indextabelle abgelegt. Dieser Vorgang wird mit der Variablenkopie und anschließender Dekrementierung wiederholt. Durch die oben beschriebenen Voraussetzungen der Ähnlichkeitsfunktion ist sichergestellt, daß sowohl der Inkrementierungs- als auch der Dekrementierungsvorgang terminieren, da die berechneten Ähnlichkeitswerte an den geforderten Grenzwerten a und b definitiv unter die Mindestähnlichkeit s fallen werden.

Die Suchanfrage nach Datensätzen, deren ganzzahliges Attribut eine angegebene Mindestähnlichkeit zu einem Suchwert aufweist, erfolgt nun über die Indextabelle *intidx*. Zusätzlich ist ein Verbund mit der Datentabelle notwendig, der die Attribute der relevanten Datensätze liefert, da die Indextabelle nur Referenzen auf die Primärschlüssel der Tupel verwaltet.

Das Inkrement bzw. Dekrement für den oben beschriebenen Vorgang ist dabei anwendungsspezifisch zu wählen. Hätte beispielsweise ein Attribut einen Wertebereich von eins bis 10 Millionen und nähme die Ähnlichkeit bei einer Abweichung von eins um den Wert 0,005 ab, so wäre es wenig sinnvoll, beim Aufbau der Indexdaten den Attributwert schrittweise um eins zu erhöhen bzw. zu erniedrigen, da dies zur Folge hätte, daß die Indexstruktur pro Datensatz vierhundert Einträge zu verwalten hätte. Hier böte es sich an, den Attributwert um vielleicht 25 zu erhöhen bzw. zu erniedrigen, und anschließend bei der Suche die Suchwerte an die so entstandenen Indexdaten anzupassen. Wünscht der Benutzer die Ausgabe einer exakten Ähnlichkeit des Suchwertes zum Attributwert, so kann diese nachträglich berechnet werden. Die Hauptaufgabe der Indexstruktur, möglichst schnell die Ergebnismenge zu bestimmen, wird durch das grobere Inkrement bzw. Dekrement nicht eingeschränkt (siehe dazu das Beispiel in Abschnitt 5.2.4).

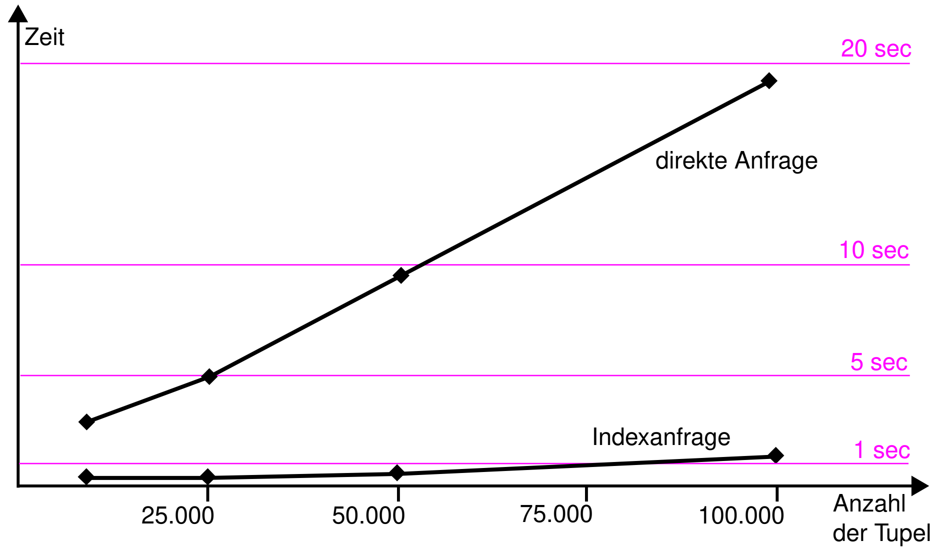
Die folgende Syntax 46 zeigt die Suchanfrage nach Datensätzen mit dem Suchwert *2000* und einer Ähnlichkeit größer *0,7*. Das Attribut *data* aus der Datentabelle *intdata* wird dabei über einen Verbund ermittelt.

Syntax 46 Suchanfrage über die Indextabelle intidx

```
SELECT i.tupleid, d.data, i.sim
FROM   intidx i, intdata d
WHERE  i.tupleid=d.id
  AND  i.value = 2000
  AND  i.sim > 0.7
ORDER BY i.sim DESC
```

Mit einer nach dieser Methode angelegten Indextabelle wurden mehrere Leistungsmessungen durchgeführt. Die Datentabelle enthielt dabei zwischen 10.000 und 100.000 Datensätzen. Die folgende Abbildung 14 zeigt die ermittelten Meßergebisse.

Abbildung 14 Leistungsmessung mit ganzzahligen Daten



Die Anfrage über die Indextabelle benötigt bei 10.000 Datensätzen nur etwa 23,2 Prozent der Zeit, die eine direkte Anfrage auf der Datentabelle erfordert. Bei 100.000 Datensätzen benötigt die Suchanfrage mit der Indextabelle nur noch etwa sechs Prozent gegenüber einer direkten Anfrage. Die genaue Durchführung und Ergebnisse der Leistungsmessungen können im Anhang B in Abschnitt B.2 nachgelesen werden.

Der Einsatz eines inversen Index für die ähnlichkeitsbasierte Suche auf ganzzahligen Datentypen ist damit zu empfehlen und kann enorme Leistungssteigerungen bewirken.

5.2.4 Inverser Index für Fließkommazahlen

Für Fließkommazahlen kann zur Konstruktion eines inversen Index die gleiche Idee wie beim Aufbau eines Indizes für ganzzahlige Daten verwendet werden.

Gemäß dem Wertebereich des Fließkommaattributs ist ein Faktor zu wählen, der die Werte (mit anschließender Rundung) auf einen möglichst großen ganzzahligen Bereich abbildet. Entsprechend dieses Bereichs ist zur Vorberechnung der Ähnlichkeitswerte ein Inkrement bzw. Dekrement zu bestimmen, das auf der einen Seite genügend Flexibilität bei der Ähnlichkeitsberechnung zuläßt, auf der anderen Seite die Menge der Indexdaten nicht zu groß werden läßt.

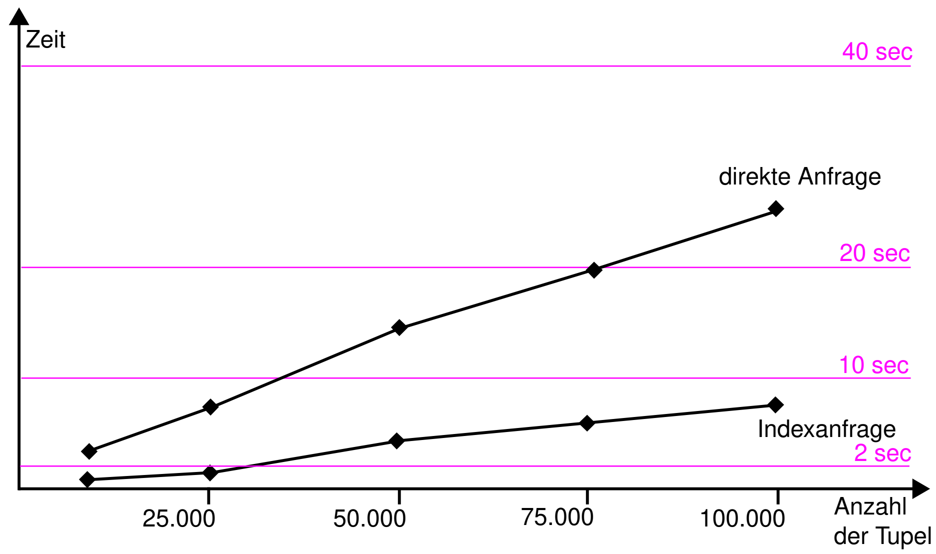
Bei der Suchanfrage muß zunächst der Suchwert mit dem zuvor bestimmten Faktor und anschließender Rundung auf den Wertebereich der Indexdaten abgebildet werden, danach kann eine Ähnlichkeitssuche auf den Indexdaten durchgeführt werden. Wird vom Benutzer der exakte Ähnlichkeitswert des Datensatzes mit der von ihm spezifizierten Vergleichsinstanz benötigt, so kann anschließend ein Aufruf der Ähnlichkeitsfunktion mit den durch die Indexdaten qualifizierten Datensätzen erfolgen.

Allerdings muß beachtet werden, daß das Suchergebnis durch die Rundung und Wahl eines entsprechenden Inkrements bzw. Dekrements ungenau wird. Im Suchergebnis können Datensätze auftreten, deren Ähnlichkeit nur im Bereich der spezifizierten Mindestähnlichkeit liegt, diese aber nicht erfüllt. Durch eine exakte Berechnung der Ähnlichkeit können die fälschlich aufgenommenen Datensätze nachträglich ausgeschlossen werden. Ebenso können Datensätze nicht in der Ergebnismenge vorkommen, wenn der aus Rundungen entstandene zu indexierende Attributwert eine Ähnlichkeit besitzt, die die Mindestähnlichkeit knapp verfehlt.

Beispiel. Wir gehen von einem Attribut *a* eines Datensatzes aus, dessen Wertebereich zwischen *0* und *1* liegt und das die Werte *0,01*, *0,02*, *0,03*, ..., *0,98*, *0,99*, *1* annehmen kann. Hier bietet es sich an, zur Indexierung mittels eines ganzzahligen Index den Faktor *100* zu wählen, der die Werte des Attributs *a* auf die ganzzahligen Indexwerte *0*, ..., *100* abbildet. Um die Anzahl der zu berechnenden Indexwerte nicht allzu groß werden zu lassen, könnte man den Indexwert des Attributs auf die nächst größere gerade Zahl erhöhen und als Inkrement bzw. Dekrement den Wert *2* wählen; somit stehen in der Indextabelle nur gerade Werte zwischen *0* und *100*. Bei einer Suchanfrage ist dann einfach der Suchwert auf eine gerade Zahl zu runden und damit eine Anfrage auf die Indextabelle durchzuführen.

Für das oben beschriebene Beispiel wurde eine Leistungsmessung durchgeführt, die Ergebisse sind in der Abbildung 15 graphisch dargestellt. Ab 50.000 Tupeln in der Datentabelle benötigt eine Anfrage über die Indextabelle noch etwa 30 Prozent der Zeit, die eine direkte Anfrage auf der Datentabelle benötigt. Der Leistungsgewinn, der durch Einsatz der Indextabelle erzielt wird, ist trotz gleichen Aufbaus der Indextabelle nicht so groß wie bei ganzzahligen Daten, da für jeden Datensatz der Ergebnismenge noch ein Funktionsaufruf der Ähnlichkeitsfunktion erfolgt, der den exakten Ähnlichkeitswert zum Suchwert berechnet. Das bedeutet, je kleiner die Ergebnismenge ist, desto größer ist der zu erwartende Leistungsgewinn bei Einsatz einer Indextabelle.

Abbildung 15 Ausführungszeiten für Fließkommadaten



Die genaue Durchführung und Ergebnisse der Leistungsmessung für Fließkommadaten sind im Anhang B im Abschnitt B.4 nachzulesen.

5.2.5 Inverser n-gram Index für Zeichenketten

Zur Unterstützung der Ähnlichkeitssuche auf Zeichenketten mit einem inversen Index wird in diesem Abschnitt das Verfahren der n-gram Technik (siehe Abschnitt 2.2.7) zur Ähnlichkeitsberechnung verwendet.

Eine Ähnlichkeitsfunktion berechnet dafür den Ähnlichkeitswert zweier Strings nach folgendem Prinzip. Sowohl der Wert des Attributs als auch der Suchwert werden in Trigrams (Teilzeichenketten der Länge 3) zerlegt. Der Quotient aus gemeinsamen Trigrams und insgesamt vorhandenen Trigrams bildet den Ähnlichkeitswert der beiden Zeichenketten.

Beim direkten Zugriff auf die Datentabelle muß die Ähnlichkeitsfunktion für jede Instanz eine Zerlegung des Attributwertes und des Suchwertes in ihre Trigrams vornehmen. Da die Zerlegung des Suchwertes bei jedem Funktionsaufruf identisch durchgeführt werden muß, bietet es sich aus Leistungsgründen an, den Suchbegriff bereits als Zerlegung in Trigrams zu übergeben. Die folgende Syntax 47 zeigt eine Suchanfrage auf der Datentabelle *ngramdata* nach dem Begriff *eintrag*.

Syntax 47 Suchanfrage auf der Datentabelle

```
SELECT id, name
FROM   ngramdata
WHERE  simTrigram (name, 'ein,int,ntr,tra,rag') > 0.2
```

Problematisch ist hier, daß die Ähnlichkeitsfunktion *simTrigram()* für den Vergleich des Suchwertes mit jedem Datensatz aufgerufen werden muß. Zusätzlich arbeitet die Ähnlichkeitsfunktion mit der Zerlegung und dem Vergleich von Zeichenketten, was zusätzlich hohe Kosten verursacht, und bei einer durchgeführten Leistungsmessung mit 10.000 Datensätzen bereits eine Laufzeit von fast zehn Sekunden verursacht hat.

Die Idee für eine Indextabelle beruht hier auf der Extraktion und Speicherung von Trigrams, so daß die Ähnlichkeitsfunktion nicht für jede Instanz aufgrufen werden muß, sondern direkt aus der Indextabelle ermittelt werden kann, welche Datensätze eine vorgegebene Menge von Trigrams beinhalten. Syntax 48 zeigt den Aufbau einer Indextabelle für Trigrams.

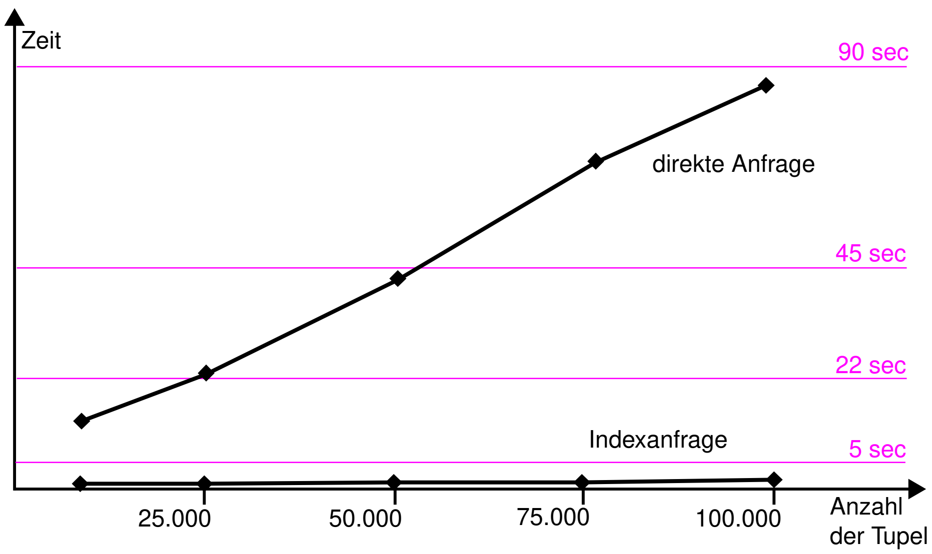
Syntax 48 Indextabelle ngramidx

```
CREATE TABLE ngramidx
(
  trigram   char(3),
  dataid    int      REFERENCES ngramdata(id)
);
```

Das zu indexierende Attribut jedes Datensatzes wird in seine Trigrams zerlegt und diese zusammen mit einer Referenz auf den Datensatz in der Indextabelle *ngramidx* abgelegt. Ein zusätzlich angelegter Index auf dem Attribut *trigram* der Indextabelle beschleunigt den Zugriff.

Nun kann mit einer Anfrage über die Indextabelle sehr schnell ermittelt werden, welche Datensätze wieviele Trigrams des Suchbegriffs enthalten. Die Gesamtzahl der Trigrams eines Datensatzes kann mit einer benutzerdefinierten Funktion ebenfalls sehr schnell aus der Indextabelle bestimmt werden. Um den Leistungsgewinn dieser Indextechnik beurteilen zu können, wurden einige Messungen durchgeführt, die folgende Abbildung 16 zeigt das Resultat.

Abbildung 16 Leistungsmessung mit Zeichenketten



Die Ausführungszeit der Suchanfrage steigt bei 100.000 Datensätzen auf fast eineinhalb Minuten an, während der Zugriff über die Indextabelle das gleiche Ergebnis in etwa 1,5 Sekunden liefert. Selbst bei 10.000 Datensätzen benötigt die Anfrage über die Indextabelle nur etwa 5,5 Prozent der direkten Zugriffszeit mit Aufruf einer Ähnlichkeitsfunktion. Bei 100.000 Tupeln beträgt die Ausführungszeit nur noch etwa 1,6 Prozent. Die genauen Meßdaten sind im Anhang B in Abschnitt B.5 erläutert.

Da zu jedem Datensatz im allgemeinen mehrere Trigrams in der Indextabelle gespeichert werden, bietet es sich an, objekt-relationale Elemente einzusetzen und die Trigrams in einem mehrwertigen Attribut (*Collection*) zu speichern. Messungen haben jedoch ergeben, daß ein Indexzugriff über ein mehrwertiges Attribut schon bei 25.000 Datensätzen fast die sechsfache Zeit eines direkten Zugriffs auf der Datentabelle benötigt (siehe Anhang B, Abschnitt B.5.4). Die Verwendung objekt-relationaler Technologie ist daher in diesem Zusammenhang nicht weiter zu verfolgen.

Der Einsatz eines inversen Indizes für die ähnlichkeitsbasierte Suche bringt wie in Abbildung 16 zu erkennen enorme Leistungsgewinne, er sollte daher unbedingt verwendet werden. Eine Ausführungszeit von fast 45 Sekunden bei 50.000 Datensätzen verbietet den Einsatz eines Ähnlichkeitsfunktionsaufrufs, da dieser zu einem unzumutbarem Verhalten des Systems führt.

5.2.6 Function-value-indexing

In diesem Abschnitt wollen wir noch kurz die Methode des *Function-value-indexing* zur Beschleunigung des Aufrufs einer Ähnlichkeitsfunktion betrachten.

Function-value-indexing ([IFMXSQLa01]) bedeutet die Indexierung von Tupeln einer Tabelle; als Schlüssel wird dabei nicht der Wert eines Tabellenattributs verwendet, sondern der Wert wird von einer Funktion berechnet, die als Parameter nur Attribute der zu indexierenden Tabelle erhalten darf. Nachdem ein Index mit der *Function-value-indexing* Technik erstellt worden ist, wird nach dem Einfügen eines neuen Tupels in die Tabelle der Index mit dem durch die Indexfunktion berechneten Schlüssel erweitert. Sobald nun ein Funktionsaufruf mit entsprechenden Attributen in einer SQL Anfrage erfolgt, muß nicht für jedes Tupel der Tabelle ein Funktionswert errechnet werden, sondern der *Function-value-index* kann zur Bestimmung der qualifizierten Tupel herangezogen werden.

Die Leistungsvorteile, die durch das *Function-value-indexing* zu erzielen sind, wurden mit mehreren Messungen bei einer Suchanfrage auf einem ganzzahligen Attribut untersucht (siehe dazu im Anhang B den Abschnitt B.3).

Die Suche auf einem ganzzahligen Attribut mit Einsatz der *Function-value-indexing* Technik wird etwa in der gleichen Zeit wie eine Suchanfrage über den in Abschnitt 5.2.3 beschriebenen Indexzugriff durchgefüht. Allerdings sind hierbei starke Einschränkungen vorhanden, die den Einsatz einer solchen Methode nur sehr bedingt erlauben.

Zur Erläuterung gehen wir von einer Datentabelle *intdata* aus, in der jeder Datensatz aus einer eindeutigen ID und einem ganzzahligen Datenattribut *data* besteht.

Syntax 49 Datentabelle intdata

```
CREATE TABLE intdata
(
  id    int  PRIMARY KEY,
  data  int  NOT NULL
);
```

Um nun auf dieser Tabelle eine ähnlichkeitsbasierte Suche mit Unterstützung des *Function-value-indexing* durchzuführen, benötigen wir eine Ähnlichkeitsfunktion, mit deren Aufruf wir einen Index anlegen können. Das Problem besteht nun darin, daß diese Funktion als Parameter nur Attribute der Tabelle *intdata* erhalten darf, und somit der Suchwert fest in den Funktionsrumpf implementiert werden muß.

Sei also *simInt2000(value)* eine Ähnlichkeitsfunktion, die die Ähnlichkeit des Wertes *2000* zum übergebenen Attributwert *value* berechnet. Dann wird der Index *funcValueIdx2000* für die Tabelle *intdata* nach der folgenden Syntax 50 angelegt.

Syntax 50 Indexdefinition

```
CREATE INDEX funcValueIdx2000 ON intdata (simInt2000(data));
```

Eine Suchanfrage nach dem Suchwert *2000* mit Aufruf der Funktion *simInt2000()* benötigt etwa die gleiche Zeit, die eine Anfrage über die in Abschnitt 5.2.3 definierte Indexstruktur erfordert. Allerdings gilt dies nur bei einer Suche nach dem Wert *2000* und der zuvor bekannten Anfrage mit dem festkodierten Funktionsaufruf von *simInt2000()*. Soll wie bei dem Zugriff über die Indexstruktur die Suche nach einem beliebigen Wert möglich sein, so muß zumindest für jeden der häufiger zu suchenden Werte ein Index angelegt werden und in Abhängigkeit des zu suchenden Wertes eine Suchanfrage mit Aufruf der entsprechenden Ähnlichkeitsfunktion generiert werden. Ist keine passende Ähnlichkeitsfunktion für den Suchwert vorhanden, so sollte die Suche auf jeden Fall über eine Indextabelle erfolgen, da hier, wie oben beschrieben, die Anfragezeit auf bis zu sechs Prozent der direkten Zugriffszeit reduziert werden kann.

Da jedoch selbst bei 100.000 Tupel die Anfrage mittels *Function-value-indexing* nicht schneller als eine Indexanfrage erfolgt, ist mit dem zusätzlich nötigen Aufwand der dynamischen Generierung der Suchanfrage in Abhängigkeit des Suchwertes kein Leistungsvorteil vorhanden.

In Abschnitt B.3 wurde zusätzlich noch eine Messung für den Einsatz des *Function-value-indexing* bei taxonomischen Daten durchgeführt. Dabei ergab sich sogar ein klarer Nachteil des Verfahrens. Ein direkter Zugriff auf die Datentabelle oder ein Zugriff über die in Abschnitt 5.2.2 eingeführte Indexstruktur wird deutlich schneller ausgeführt.

Das *Function-value-indexing* bietet daher keine Vorteile zur Unterstützung einer ähnlichkeitsbasierten Suche in einem objekt-relationalen Datenbanksystem.

5.2.7 Aggregation

Um für die Aggregation mehrerer lokaler Ähnlichkeitsmaße (siehe dazu auch Abschnitt 2.2.9) die separat verwalteten lokalen Indexdaten nutzen zu können, muß ein Verbund über die Primärschlüssel der Datensätze aus den einzelnen Indexdaten durchgeführt werden. Dies ist für alle in den obigen Abschnitten behandelten Datentypen möglich, da bei jeder Suchanfrage über die lokalen Indexdaten der Primärschlüssel als mögliches Verbundattribut mit zurückgeliefert wird. Der Verbund ist aus Leistungsgründen dem Funktionsaufruf einer Ähnlichkeitsfunktion vorzuziehen.

Die Art des zu verwendenden Verbundes ist von der gewünschten Semantik des Anwenders abhängig. Sollen alle spezifizierten Attribute der Suche eine lokale Ähnlichkeit größer einer lokalen Mindestähnlichkeit aufweisen, so kann ein innerer Verbund (*inner join*) durchgeführt werden, da für jeden qualifizierten Datensatz ein Eintrag in allen lokalen Indexstrukturen vorliegt. Sollen auch Datensätze ausgegeben werden, deren Attribute eine lokale Ähnlichkeit unter einer lokalen Mindestähnlichkeit besitzen, die Gesamtähnlichkeit aber dennoch eine geforderte globale Mindestähnlichkeit erfüllt, so ist ein äußerer Verbund (*outer join*) anzuwenden, da für einen Datensatz nur in den Indexstrukturen mit lokaler Ähnlichkeit größer einem lokalen Mindestwert ein Eintrag vorliegt, und die restlichen Angaben durch *NULL*-Werte ergänzt werden müssen.

Die so ermittelten lokalen Ähnlichkeitswerte werden dann mit der vom Benutzer angegebenen Aggregationsfunktion entsprechend der vorgegebenen Gewichte zu der globalen Gesamtähnlichkeit verrechnet. Syntax 33 in Anhang B im Abschnitt B.6.2 zeigt ein Beispiel für eine solche Suchanfrage über drei lokale ganzzahlige Indextabellen, deren jeweilige Ähnlichkeitswerte mit individuellen Gewichten zu einer Gesamtähnlichkeit verrechnet werden.

5.2.8 NULL-Werte

Um die Ähnlichkeitssuche mit lokalen Ähnlichkeitsmaßen zu unterstützen, können nach Abschnitt 5.2.2 bis Abschnitt 5.2.5 inverse Indizes für ganzzahlige Daten, Fließkommadaten und Zeichenketten eingesetzt werden. Für Taxonomien bringt der Einsatz eines inversen Index keine Vorteile.

Beim Aufbau eines inversen Indizes für ganzzahlige Daten und Fließkommadaten werden durch Inkrementierung bzw. Dekrementierung des Attributwertes Ähnlichkeitswerte vorberechnet und in einer Indextabelle abgelegt. Zeichenkettenattribute werden in N-grams zerlegt und diese in der Indextabelle zusammen mit einer Referenz zum Datensatz gespeichert.

Nehmen die zu indexierenden Attribute nun NULL-Werte an, so kann eine Berechnung der Indexdaten nicht mehr durchgeführt werden, und ein Auffinden der Datensätze allein über die Indexstruktur ist nicht möglich. Daher hat die Aggregation der lokalen Ähnlichkeiten zur globalen Gesamtähnlichkeit implizit die Semantik, daß ein auf NULL gesetzter Attributwert mit dem lokalen Ähnlichkeitswert 0 in die Berechnung eingeht.

5.3 Keine Parametrisierung

Bei der ähnlichkeitsbasierten Suche ohne Parametrisierung kann der Benutzer für seine Suchanfrage keinerlei Parameter einstellen, um damit das Suchergebnis zu beeinflussen.

5.3.1 Eigenschaften

Bei einer Suche ohne Parameter sind die Möglichkeiten des Anwenders am stärksten eingeschränkt. Das lokale Ähnlichkeitsmaß für jedes Attribut einer Tabelle ist fest vorgegeben, keine der Funktionen kann durch Parameter angepaßt werden. Die Gewichte der einzelnen Attribute, die die Relevanz eines Attributs bei der Suche ausdrücken, sind fest eingestellt. Ebenso ist die Aggregationsfunktion, die aus den lokalen Ähnlichkeitsmaßen und den Gewichten eine globale Gesamtähnlichkeit berechnet, fest vorgegeben und kann nicht mehr verändert werden.

In objekt-relationalen Datenbanksystemen, die es erlauben, Tabellen in Vererbungshierarchien zu organisieren, wird für jede Tabelle in der Hierarchie ein fester Ähnlichkeitsfaktor definiert, mit dem eine *Intra-Klassenähnlichkeit* zwischen Tupeln verschiedener Tabellen berechnet werden kann (siehe Abschnitt 2.3).

Der Benutzer kann an der Ad-hoc Schnittstelle des Datenbanksystems eine Anfrage, die ähnlichkeitsbasiert ausgewertet wird, ohne die Angabe von Parametern stellen. Alternativ kann vom System eine Suchmaske angeboten werden, in der der Anwender eine Vergleichsinstanz durch die Angabe von Attributwerten spezifizieren kann.

5.3.2 Unterstützung mit einem globalen inversen Index

Die ähnlichkeitsbasierte Suche ohne Parametrisierung kann durch einen globalen inversen Index unterstützt werden. Dieser Index kann für alle taxonomischen und ganzzahligen Attribute sowie Fließkommaattribute aufgebaut werden, da für diese Attribute in den lokalen inversen Indizes (siehe Abschnitt 5.2) Such- und Ähnlichkeitswerte mit Referenzen zu den entsprechenden Datensätzen vorhanden sind.

Für Zeichenkettenattribute kann die Berechnung einer Ähnlichkeit erst zum Zeitpunkt der Suchanfrage durchgeführt werden, da zur Ermittlung des Ähnlichkeitswertes der konkrete Suchbegriff erst vorliegen muß (siehe Abschnitt 5.2.5).

Zur Erzeugung der Daten des globalen inversen Index wird dabei nach folgendem Verfahren vorgegangen. Wird ein Datensatz in eine zu indexierende Tabelle eingefügt oder aktualisiert, so werden für jedes Attribut entsprechende Indexdaten in den lokalen inversen Indextabellen erzeugt, die die Suche auf dem jeweiligen Attribut unterstützen. Besteht also ein Datensatz beispielsweise aus den Attributen a, b und c und hat die Gestalt (a, b, c), so werden beim Einfügen oder Aktualisieren die lokalen Indexdaten $a_1,...,a_m$, $b_1,...,b_n$ und $c_1,...,c_p$ erstellt.

Für diese Daten müssen im nächsten Schritt alle möglichen Kombinationen (a_x, b_y, c_z) mit $a_x \in \{a_1, \ldots, a_m\}$, $b_y \in \{b_1, \ldots, b_n\}$ und $c_z \in \{c_1, \ldots, c_n\}$ bestimmt werden. Für jede Wertekombination wird nun mit der fest vorgegebenen Aggregationsfunktion die globale Gesamtähnlichkeit berechnet. Ist diese größer gleich der festgelegten globalen Mindestähnlichkeit, so wird das Wertetupel zusammen mit dem berechneten Ähnlichkeitswert und einer Referenz zum Datensatz in die globale Indextabelle aufgenommen.

Das Verfahren läßt sich in der oben beschriebenen Variante nur anwenden, wenn der Benutzer bei jeder Suchanfrage alle Attribute der Datentabelle spezifizieren muß. Soll der Anwender nicht für jedes Attribut einen Vergleichswert angeben müssen, so muß zur Bestimmung der Wertekombinationen, für die ein globaler Ähnlichkeitswert berechnet wird, in die Liste der lokalen Indexdaten für jedes Attribut noch ein NULL-Wert aufgenommen werden. Dieser NULL-Wert bewirkt bei der Vorberechnung der globalen Gesamtähnlichkeit, daß für dieses Attribut kein Vergleichswert spezifiziert werden muß.

Da jede Suchanfrage mit einem Zugriff auf den globalen inversen Index ausgeführt werden kann, ist es nicht notwendig, die zur Vorberechnung der globalen Gesamtähnlichkeit erzeugten lokalen Indexdaten zu speichern.

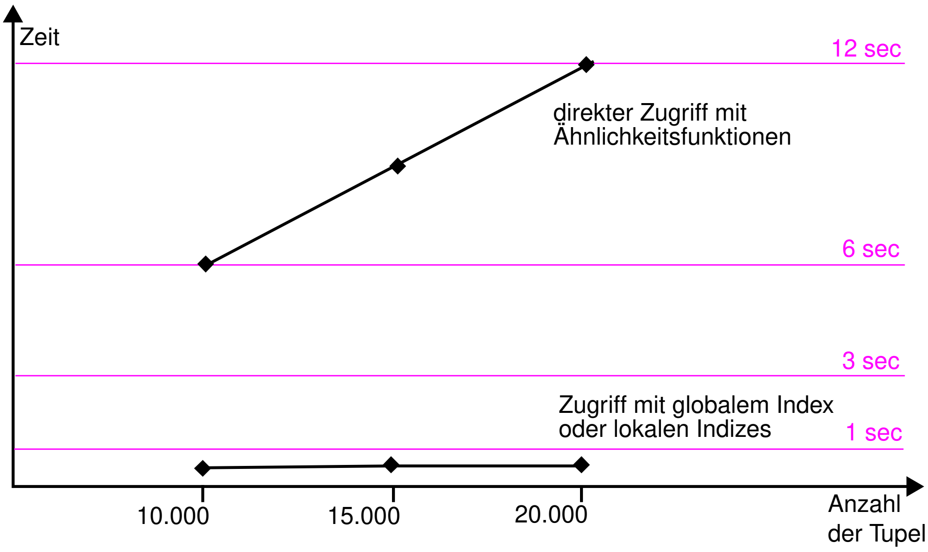
Der globale inverse Index ist für jede Tabelle separat zu verwalten. Da auch der Faktor zur Berechnung der *Intra-Klassenähnlichkeit* fest vorgegeben ist, kann der Ähnlichkeitswert einer Wertekombination zu einem Datensatz einer anderen Tabelle vorberechnet werden und in deren globalen inversen Index eingetragen werden. Somit werden bei einer Suchanfrage auf den globalen Index einer Tabelle auch Tupel anderer Tabellen zurückgeliefert, die sich über die *Intra-Klassenähnlichkeit* qualifizieren.

5.3.3 Leistungsbewertung

Zur Beurteilung des Leistungsverhaltens bei Einsatz eines globalen Indizes wurden mehrere Messungen durchgeführt. Wir vergleichen dazu bei Suchanfragen auf einer Tabelle mit mehreren Attributen die Antwortzeiten bei Zugriff über den globalen Index, Zugriff mit einem Verbund über die lokalen Indizes und bei direktem Zugriff auf die Datentabelle mit Verwendung von Ähnlichkeitsfunktionen. Die folgende Abbildung 17 zeigt die Ergebnisse der Messung.

Wie in der Graphik zu erkennen ist, bringt der Einsatz eines globalen Index gegenüber dem direkten Zugriff auf die Datentabelle starke Leistungsvorteile. Bei einer Datenmenge von 20.000 Tupeln benötigt eine Suchanfrage sowohl über die globale Indexstruktur als auch über die lokalen Indexdaten nur noch etwa 3,4 Prozent der Zeit, die zur direkten Suche auf der Datentabelle mit Aufrufen der Ähnlichkeitsfunktionen nötig ist. Die genauen Ergebnisse und Durchführung der Messung können im Anhang B im Abschnitt B.6 nachgelesen werden.

Abbildung 17 Leistungsmessung mit globalem Index



Allerdings läßt sich diese Leistungssteigerung auch mit dem Zugriff über die lokalen Indexdaten erreichen, jedoch mit dem weiteren Vorteil, daß hier mehr Freiheiten bezüglich der Parameterwahl bestehen. Da nur die lokalen Indexdaten verwendet werden, und die globale Ähnlichkeit erst innerhalb der Suchanfrage berechnet wird, handelt es sich dabei um die in Abschnitt 5.2 eingeführte *schwache Parametrisierung*.

Der Einsatz eines globalen Index als Folge der parameterlosen Ähnlichkeitssuche bringt gegenüber der schwachen Parametrisierung mit Verwendung lokaler Indexdaten keine Leistungsvorteile. Da dem Benutzer in Folge dessen auch die Möglichkeit der schwachen Parametrisierung genommen wird, ist der Einsatz eines globalen Index nicht zu empfehlen.

5.4 Zusammenfassung

Die starke Parametrisierung ermöglicht dem Benuzter die maximale Freiheit, das Suchverfahren an seine persönlichen Bedürfnisse anzupassen. Allerdings ist außer einer Datenpufferung bei der Suchauswertung eine Optimierung der Suchanfrage durch eine Indexstruktur nicht möglich und führt daher zu einem nicht zumutbaren Systemverhalten.

Bei der schwachen Parametrisierung entfällt die Möglichkeit, die lokalen Ähnlichkeitsmaße zu modifizieren. Dadurch kann die Suchanfrage jedoch durch lokale Indexstrukturen stark beschleunigt werden.

Der völlige Verzicht auf Suchparameter schränkt den Benutzer unnötig ein, da keine Leistungsvorteile mehr erbracht werden. Wie auch im Abschnitt 5.2.2 eingeführten inversen Index für Taxonomien ist hier zu erkennen, daß ein Index keine Leistungsvorteile erzielt, wenn die Ähnlichkeitswerte bereits in Tabellen im Datenbanksystem vorliegen und über einen Verbund bestimmt werden können. In diesem Fall führt der Einsatz eines Index nur zu einem erhöhten Datenvolumen aufgrund der zu speichernden Indexdaten.

Indizes bringen daher nur Leistungsvorteile, wenn durch den Zugriff über die Indexstrukturen Funktionsaufrufe von Ähnlichkeitsmaßen eingespart werden können.

Kapitel 6 Zusammenfassung und Ausblick

Zum Abschluß dieser Arbeit wollen wir den Inhalt der zuvor stehenden Kapitel noch einmal kurz zusammenfassen und einige Aspekte ansprechen, die im Zusammenhang mit der ähnlichkeitsbasierten Suche noch näher untersucht werden sollten.

6.1 Zusammenfassung

Nach einer kurzen Einleitung in Kapitel 1 haben wir uns in Kapitel 2 den Ähnlichkeitsmaßen zugewandt. Wir haben zahlreiche Möglichkeiten beschrieben, in Abhängigkeit der Art der Datenrepräsentation Ähnlichkeitswerte zu berechnen. Als Folge dieser Erkenntnis stellten wir einige Ähnlichkeitsmaße für Attribut-Wert-, objekt-orientierte-, Graph- und prädikatenlogische Datenrepräsentationen vor. Besonders hervorzuheben ist das Lokal-Global-Prinzip in Abschnitt 2.2.3, durch das das Zusammenspiel von lokalen und globalen Ähnlichkeitsmaßen zur Berechnung eines Gesamtähnlichkeitswertes eingeführt wird.

Kapitel 3 beschäftigt sich mit den vorhandenen Ansätzen der ähnlichkeitsbasierten Suche beim Text-Retrieval in objekt-relationalen Datenbanksystemen drei bekannter Hersteller. Hier wurden die Produkte *Text Extender*, *Net Search Extender*, *Text Information Extender* und *Excalibur Text Search DataBlade* von *IBM* bzw. *IBM Informix* für die Datenbanksysteme *DB2* bzw. *Informix* und *Oracle 9iText* der Firma *Oracle* vorgestellt. Ähnlichkeitssuche in diesem Zusammenhang beschränkt sich aber auf die Suche in zuvor indexierten Dokumentenmengen unter dem Einsatz von Stopwortlisten und Thesauri. Das bedeutet zum einen, daß die Suche nur auf einer Tabellenspalte, die die Textdokumente verwaltet, stattfindet. Zum anderen sind Ähnlichkeiten zuvor von Anwendern manuell spezifiziert worden und eine Ähnlichkeitsberechnung, wie in Kapitel 2 vorgestellt, findet hier nicht statt.

In Kapitel 4 haben wir Forderungen zusammenstellt, die ein objekt-relationales Datenbanksystem, das eine ähnlichkeitsbasierte Suche anbietet, erfüllen sollte. Nach einer kurzen Beschreibung der Möglichkeiten, die ein objekt-relationales Datenbanksystem zur Datenmodellierung anbietet, fordern wir die Berechnung lokaler Ähnlichkeiten zwischen einzelnen Attributwerten. Diese sollten schließlich mittels Gewichtung und wählbarer Funktionen zu einer Gesamtähnlichkeit beliebiger Datensatztypen aggregierbar sein. Werden objekt-relationale Elemente zur Datenmodellierung eingesetzt, so sollte die Objekthierarchie mit in die Ähnlichkeitsberechnung einfließen.

In Kapitel 5 diskutieren wir schließlich einige Realisierungsaspekte einer Ähnlichkeitssuche in objekt-relationalen Datenbanksystemen. Dabei unterscheiden wir zwischen starker und schwacher Parametrisierung und einer Suche ohne parametrisierbare Funktionen. Wir geben für jeden der genannten Fälle Möglichkeiten der Realisierung einer Suche an und stellen Indexstrukturen vor, durch die eine Suchanfrage schneller abgearbeitet werden kann. Die Leistungsgewinne wurden durch zahlreiche Messungen belegt. Generell kann durch den Einsatz einer Indexstruktur mit vorberechneten Ähnlichkeitswerten ein enormer Leistungsgewinn erzielt werden, wenn die Ermittlung von Ähnlichkeitswerten aus den Datentabellen durch Aufrufe von Ähnlichkeitsfunktionen erfolgt. Liegen die Ähnlichkeitswerte jedoch bereits in Tabellen im Datenbanksystem vor und werden durch einen Verbund ohne Aufruf eines Ähnlichkeitsmaßes bestimmt, so bringt ein Index keine Leistungsvorteile.

Zur Realisierung einer ähnlichkeitsbasierten Suche kann auf objekt-relationale Konzepte des Datenbanksystems nicht verzichtet werden. Die Indexdaten müssen bei Einfüge-, Lösch- und Aktualisierungsvorgängen in den Datentabellen stets gewartet werden. Dazu müssen *Trigger* ([SQL99a], [SQL99b]) definiert werden, die bei einer der oben genannten Operationen eine Aktualisierung der Indextabellen anstoßen. Für Berechnungen von Ähnlichkeitswerten zur direkten Ausgabe an den Anwender oder zur Erzeugung von Indexdaten sind benutzerdefinierte Funktionen sehr hilfreich. Sie ermöglichen die Auswertung von Ähnlichkeitsmaßen ohne Kommunikation mit einer externen Anwendung und das einfache Austauschen eines Ähnlichkeitsmaßes durch die gegebene Funktionskapselung. Die Konzepte zur Organisation von Daten in Tabellenhierarchien mit komplexen Datentypen ermöglichen eine sehr genaue Anpassung der Datenstruktur an das Anwendungsproblem und erlauben eine direkte Realisierung der Ähnlichkeitssuche auf dieser Struktur ohne Implementierung einer Abbildungsschicht. Auf den Einsatz komplexer Datentypen sollte jedoch aufgrund der recht komplizierten Definitions- und Anfragesyntax, sowie der schlechten Leistungseigenschaften weitgehend verzichtet werden.

6.2 Ausblick

Zur Realisierung einer ähnlichkeitsbasierten Suche ist ein Konzept zur Integration in die Standardsprache *SQL* für die Indexierung, Suchanfrage und Anfrageabbildung zu entwerfen.

- Es muß die Möglichkeit geschaffen werden, über Befehle (Erweiterung der Standardsprache SQL) lokale Indexstrukturen für einzelne Attribute anlegen zu können. Dabei müssen die durch den Index unterstützten Ähnlichkeitsmaße und dafür benötigte Parameter bei der Indexdefinition mit angegeben werden.
- Weiterhin muß der Anwender die Möglichkeit haben, Ähnlichkeitsanfragen in SQL zu formulieren. Dies könnte durch ein neues Operatorsymbol oder einen Funktionsaufruf erfolgen (siehe Syntax 51). Dabei ist auch zu klären, wie die Suchparameter für die Anfrage festgelegt werden; diese könnten in einer zentralen Komponente für jeden Benutzer verwaltet oder bei jeder Suchanfrage explizit (eventuell durch neue Schlüsselwörter) angegeben werden.

Syntax 51 Suchanfragen

```
SELECT * FROM datatable WHERE attribute ~ 100;
SELECT * FROM datatable WHERE attribute ~ 100 USING param1=5 AND param2=...;
SELECT * FROM datatable WHERE sim (attribute, 100, ‘param1=5, param2=...’);
```

- Die berechneten Ähnlichkeitswerte der qualifizierten Datensätze zur vorgegebenen Vergleichsinstanz könnten über eine Funktion oder eine Variable der Ähnlichkeitsfunktion ausgegeben werden.

Syntax 52 Ähnlichkeitswerte

```
SELECT score(),data FROM datatable WHERE attribute ~ 100;
SELECT score,data
FROM   datatable
WHERE  sim (attribute, 100, ‘param1=5, param2=...’, score);
```

- Es muß untersucht werden, wie eine konkrete Suchanfrage auf die dazugehörigen Indextabellen abgebildet werden kann, um eine kurze Antwortzeit zu erzielen. Da ein Benutzer beim Anlegen eines Indizes nichts über dessen Struktur erfährt, muß die vom Benutzer gestellte Suchanfrage in eine Anfrage auf die Indextabellen übersetzt werden, die eine beschleunigte Anfrageverarbeitung für die spezifizierten Attribute unterstützen.
- Es ist anzunehmen, daß Anpassungen in tieferen Schichten eines objekt-relationalen Datenbanksystems an die Konzepte der ähnlichkeitsbasierten Suche weitere Leistungsvorteile erzielen. Dies sollte weiter verfolgt werden.

Wie an den oben aufgeführten Punkten zu erkennen ist, muß in die Erforschung der Konzepte einer ähnlichkeitsbasierten Suche noch viel Arbeit investiert werden, um dem Anwender eine benutzerfreundliche und leistungsfähige Schnittstelle für die Ähnlichkeitssuche zur Verfügung stellen zu können.

Anhang A Suche im SFB-501-Reuse-Repository

Im *Reuse Repository*, einer Datenbank zur Verwaltung von Softwareartefakten, des Sonderforschungsbereichs 501 an der Universität Kaiserslautern ist es aufgrund der starken Parametrisierung der Suchfunktionalität nicht möglich, geeignete Indexstrukturen zu verwalten.

Trotzdem kann durch die Pufferung von Daten, die von einer Ähnlichkeitsfunktion bei jedem Aufruf benötigt werden, eine Leistungssteigerung erzielt werden. Dies wird im folgenden an einem Beispiel mit konkreten Meßwerten verdeutlicht.

A.1 Problematik

Wir betrachten zur Beschreibung der gegebenen Problematik das Attribut *area_refinement* der Erfahrungselemente, die im *Reuse Repository* abgelegt sind. Das Attribut ist vom Datentyp *integer*, und darf nur Werte annehmen, für die in der gleichnamigen Referenztabelle *area_refinement* ein Datensatz angelegt ist.

Der Benutzer bekommt als Wert des Attributs *area_refinement* auf der Anwendungsoberfläche nur die String-Bezeichnung des Datensatzes der Referenztabelle zu sehen und weiß nicht, welcher Zahlenwert sich dahinter verbirgt. Der Zahlenwert ist vollkommen unabhängig von der Bedeutung des Datensatzes in der Referenztabelle, so daß Ähnlichkeiten zwischen jeweils zwei Werten des Attributs *area_refinement* manuell definiert werden müssen.

Zum einen ist es die Aufgabe des *Reuse Repository* Administrators, die Ähnlichkeitsbeziehungen zwischen den Werten des Attributs *area_refinement* anzugeben, zum anderen kann jeder Benutzer diese Vorgaben durch eigene Angaben überschreiben.

Die Ähnlichkeitsfunktion *simSelect()*, die für die Berechnung eines Ähnlichkeitsmaßes zwischen zwei Werten des Attributs *area_refinement* zuständig ist, muß nun bei jedem Aufruf überprüfen, ob eine Benutzerangabe für diese beiden Werte vorliegt. Ist dies nicht der Fall, so wird nach einem Eintrag des Administrators für diese beiden Werte gesucht. Endet auch diese Anfrage erfolglos, so wird ein fest implementierter Standardwert für die Ähnlichkeitsberechnung herangezogen.

Für zwei Attributwerte, für die weder vom Administrator noch vom Benutzer eine Ähnlichkeitsangabe gemacht wurde, werden bei jedem Funktionsaufruf zwei erfolglose Datenbankzugriffe durchgeführt, bevor der fest implementierte Standardwert verwendet wird. Für zwei Attributwerte, für die nur vom Administrator ein Ähnlichkeitswert eingegeben wurde, werden ebenfalls zwei Datenbankzugriffe durchgeführt, da die erste Suche nach einem Benutzerwert immer fehlschlägt.

A.2 Einsatz einer Puffertabelle

Eine Verbesserung dieser Situation kann durch den Einsatz einer Puffertabelle erzielt werden. Die folgende Syntax zeigt die Definition der angelegten Puffertabelle *simTempInt*.

Syntax 1 Puffertabelle simTempInt

```
CREATE TABLE simTempInt
(
  userID          int,
  attribute       char(100),
  value1          int,
  value2          int,
  sim             decimal(5,2)
)
```

In der Tabelle *simTempInt* wird für jeden Benutzer der Ähnlichkeitswert verwaltet, den zwei Werte eines gegebenen Attributs annehmen. Diese Informationen können zur Beschleunigung der Ähnlichkeitsberechnung durch die Funktion *simSelect()* verwendet werden.

Die Funktion *simSelect()* prüft in einem ersten Datenbankzugriff, ob für das angegebene Attribut und die beiden zu vergleichenden Werte bereits ein Eintrag in der Tabelle *simTempInt* vorliegt. Ist dies der Fall, so wird die dort vorliegende Ähnlichkeit zurückgeliefert. Findet sich in der Tabelle *simTempInt* kein passender Eintrag, so wird die Ähnlichkeit wie bisher berechnet, danach aber der berechnete Wert in der Tabelle *simTempInt* für den nächsten Aufruf vermerkt.

Nach der Ausführung der Suchanfrage werden die erzeugten Daten aus der Tabelle *simTempInt* wieder gelöscht, da der Benutzer vor dem Ausführen der nächsten Anfrage die Möglichkeit hat, die Suchparameter für das entsprechende Attribut zu modifizieren. Alternativ können die temporären Daten auch erst nach Änderung der Suchparameter entfernt werden.

A.3 Meßergebnisse

Das *Reuse Repository* des Sonderforschungsbereichs 501 wurde auf dem *Informix Internet Foundation.2000* Datenbankserver von *IBM Informix* implementiert und enthält zum Zeitpunkt der Messung 664 Erfahrungselemente, auf denen Suchanfragen nach dem Attribut *area_refinement* gestellt werden. Um einen repräsentativen Durchschnittswert zu erhalten, werden insgesamt fünf Anfragen gestartet.

Da die Antwortzeit davon abhängt, wieviele Ähnlichkeitswerte individuell vom Benutzer angegeben werden (das bedeutet, ob direkt mit dem ersten Datenbankzugriff ein Ähnlichkeitswert ermittelt werden kann, oder zwei Zugriffe erforderlich sind), werden zwei Messung durchgeführt.

Bei der ersten Messung ist für alle möglichen Wertepaare (Suchwert und Datensatzwert) eine Ähnlichkeit angegeben. Die zweite Messung zielt auf eine anwendungstypischere Situation. Hier sind die Ähnlichkeitsangaben des Benutzers so gewählt, daß für ein Drittel der Instanzen die Ähnlichkeit mit dem ersten Datenbankzugriff ermittelt werden kann, für zwei Drittel der Instanzen sind zwei Zugriffe erforderlich, da auf die Standardwerte des *Reuse Repository* Administrators zurückgegriffen werden muß oder die Werte keine Ähnlichkeit aufweisen.

Jede Messung wird sowohl mit als auch ohne Puffertabelle durchgeführt, die Ausführungszeiten werden notiert. Dabei wird direkt die Ähnlichkeitsfunktion *simSelect()* über das Kommandozeilenprogramm *dbaccess* aufgerufen, die Zeitmessung erfolgt, wie in Anhang B genauer nachgelesen werden kann, durch das Programm *time*.

Abbildung 1 Messung 1: Alle Ähnlichkeitswerte mit einem Zugriff ermitteln

	Anfrage ohne Puffertabelle	**Anfrage mit Puffertabelle**
Anfrage 1	36,46 sec	30,73 sec
Anfrage 2	34,85 sec	30,53 sec
Anfrage 3	34,96 sec	30,64 sec
Anfrage 4	31,82 sec	30,55 sec
Anfrage 5	34,94 sec	30,68 sec
Durchschnitt	**35,30 sec**	**30,63 sec**

Bei der derzeitig gegebenen Menge von Erfahrungselementen im *Reuse Repository* wird eine Anfrage mit Verwendung einer Puffertabelle in etwa 87 Prozent der Zeit ausgewertet, die eine Anfrage ohne Puffertabelle benötigt, wenn der Benutzer für jedes bei der Suche zu vergleichende Wertepaar eine Ähnlichkeit definiert hat.

Zwar ist sowohl mit als auch ohne Puffertabelle nur ein Datenbankzugriff zur Ähnlichkeitsbestimmung erforderlich, doch die Puffertabelle enthält weniger Datensätze als die Tabelle, die alle Ähnlichkeitswerte verwaltet, und die Anfrage ohne Puffertabelle erfordert zur Ähnlichkeitsbestimmung einen Verbund zweier Tabellen, der bei einer Puffertabellenanfrage nicht nötig ist.

Abbildung 2 Messung 2: Zwei Drittel aller Ähnlichkeitswerte mit zwei Zugriffen ermitteln

	Anfrage ohne Puffertabelle	**Anfrage mit Puffertabelle**
Anfrage 1	41,97 sec	30,48 sec
Anfrage 2	41,45 sec	30,44 sec
Anfrage 3	41,80 sec	30,76 sec
Anfrage 4	41,01 sec	30,56 sec
Anfrage 5	41,28 sec	30,43 sec
Durchschnitt	**41,50 sec**	**30,53 sec**

Werden bei zwei Drittel aller Ähnlichkeitsberechnungen zwei Datenbankzugriffe benötigt, so steigt die Antwortzeit ohne Verwendung der Puffertabelle um durchschnittlich etwa sechs Sekunden an. Die Abarbeitungszeit einer Suchanfrage mit Puffertabelle bleibt etwa gleich, das bedeutet, die anfänglich doppelten Zugriffe zum Füllen der Puffertabelle haben keine Auswirkungen auf die Antwortzeit.

In diesem Szenario benötigt eine Suchanfrage mit Verwendung der Puffertabelle nur noch etwa 74 Prozent der Zeit, die eine Suchanfrage ohne Puffertabelle erfordert.

Anhang B Leistungsmessungen mit Indizes

Um zu untersuchen, ob der Einsatz von Indizes zur Unterstützung der ähnlichkeitsbasierten Suche Leistungsvorteile erzielt, werden diverse Messungen durchgeführt. Dazu wird die Zeit beobachtet, die das System zur Verarbeitung einer Suchanfrage mit und ohne Indexdaten benötigt. Wir betrachten in den folgenden Abschnitten Messungen mit Indizes für Taxonomien, ganzzahlige Daten und Zeichenketten.

Die Leistungsmessungen werden auf einer *Sun Enterprise 450* mit vier 300 MHz Prozessoren und 1,5 Gigabyte Hauptspeicher durchgeführt. Auf dem Rechner ist das Betriebssystem *Solaris* in der Version *Solaris 8 4/01 s28s_u4wos_08 SPARC* installiert. Es wird der Datenbankserver *Informix Internet Foundation.2000* von *IBM Informix* mit der Serverversion *Informix Dynamic Server 9.30.UC1* verwendet ([IFMXSQLa01], [IFMXSQLb01]). Der vom Datenbankserver nutzbare Hauptspeicher ist nicht begrenzt.

Die Suchanfragen werden in jeweils separaten Textdateien erstellt und mit dem Kommandozeilenprogramm *dbaccess* ausgeführt. Die Zeitmessung erfolgt durch das Programm *time*, das die benötigte Zeit eines als Parameter übergebenen Befehls ausgibt. Die folgende Syntax zeigt ein Beispiel für die Zeitmessung einer einzelnen Suchanfrage (Datei *selectNgramIdx.sql*) in der Datenbank *messuredb* im Datenbankserver *idsbeta1*.

Syntax 2 Suchanfrage

```
time dbaccess messuredb@idsbeta1 selectNgramIdx.sql
```

B.1 Messungen mit lokalem Index für Taxonomien

In diesem Abschnitt führen wir die Leistungsmessungen für die in Kapitel 5 im Abschnitt 5.2.2 eingeführten lokalen Indizes für taxonomische Daten durch.

B.1.1 Datenbankschema

Zur Leistungsmessung der Ähnlichkeitsberechnung von Taxonomien wird zunächst die Tabelle *taxtypes* zur Verwaltung der taxonomischen Begriffe angelegt und darin zehn Typen eingefügt.

Syntax 3 Tabelle taxtypes mit taxonomische Begriffen

```
CREATE TABLE taxtypes
(
  id    int PRIMARY KEY,
  name  char(10)
);

INSERT INTO taxtypes (id,name) VALUES ( 1,'Type A');
INSERT INTO taxtypes (id,name) VALUES ( 2,'Type B');
INSERT INTO taxtypes (id,name) VALUES ( 3,'Type C');
INSERT INTO taxtypes (id,name) VALUES ( 4,'Type D');
INSERT INTO taxtypes (id,name) VALUES ( 5,'Type E');
INSERT INTO taxtypes (id,name) VALUES ( 6,'Type F');
INSERT INTO taxtypes (id,name) VALUES ( 7,'Type G');
INSERT INTO taxtypes (id,name) VALUES ( 8,'Type H');
INSERT INTO taxtypes (id,name) VALUES ( 9,'Type I');
INSERT INTO taxtypes (id,name) VALUES (10,'Type J');
```

Im nächsten Schritt müssen die Ähnlichkeitswerte zwischen den Taxonomietypen festgelegt werden. Dazu wird die Tabelle *taxsims* angelegt, in der die Ähnlichkeit zwischen je zwei Typen eingetragen wird.

Es werden nur Ähnlichkeiten zwischen den Typen A bis C definiert, die Typen E bis J bekommen jeweils nur einen Ähnlichkeitswert von *1* zu sich selbst zugewiesen. Dadurch erhalten wir eine übliche Situation, in der nur eine Teilmenge der Begriffe Ähnlichkeiten aufweist.

Syntax 4 Ähnlichkeitswerte in Tabelle taxsims

```
CREATE TABLE taxsims
(
  id1    int   REFERENCES taxtypes(id),
  id2    int   REFERENCES taxtypes(id),
  sim    decimal(5,2)
);

INSERT INTO taxsims (id1,id2,sim) VALUES (1,1,1);
INSERT INTO taxsims (id1,id2,sim) VALUES (1,2,0.8);
INSERT INTO taxsims (id1,id2,sim) VALUES (1,3,0.7);
INSERT INTO taxsims (id1,id2,sim) VALUES (2,1,0.8);
INSERT INTO taxsims (id1,id2,sim) VALUES (2,2,1);
INSERT INTO taxsims (id1,id2,sim) VALUES (2,3,0.95);
INSERT INTO taxsims (id1,id2,sim) VALUES (3,1,0.7);
INSERT INTO taxsims (id1,id2,sim) VALUES (3,2,0.95);
INSERT INTO taxsims (id1,id2,sim) VALUES (3,3,1);
INSERT INTO taxsims (id1,id2,sim) VALUES (4,4,1);
INSERT INTO taxsims (id1,id2,sim) VALUES (5,5,1);
INSERT INTO taxsims (id1,id2,sim) VALUES (6,6,1);
INSERT INTO taxsims (id1,id2,sim) VALUES (7,7,1);
INSERT INTO taxsims (id1,id2,sim) VALUES (8,8,1);
INSERT INTO taxsims (id1,id2,sim) VALUES (9,9,1);
INSERT INTO taxsims (id1,id2,sim) VALUES (10,10,1);
```

Nun wird die Datentabelle angelegt, auf der die Suchanfrage ausgeführt wird. Ein Datensatz besteht aus einer eindeutigen *tupleid* und einem Typ *tupletype*, der aus der Taxonomietabelle *taxtypes* bezogen wird.

Syntax 5 Datentabelle taxdata

```
CREATE TABLE taxdata
(
  tupleid     int   PRIMARY KEY,
  tupletype   int   REFERENCES taxtypes(id)
);
```

Die Datentabelle *taxdata* wird mit Daten gefüllt, die von einer benutzerdefinierten Funktion mit Hilfe eines Zufallsgenerators erzeugt werden. Dadurch können Anfragen auf große Datenmengen (25.000, 50.000 und 100.000 Tupel) durchgeführt werden.

Zur Unterstützung der Suchanfrage wird eine Indextabelle *taxidx* angelegt, in der die vorberechneten Ähnlichkeitswerte gespeichert werden.

Syntax 6 Indextabelle taxidx

```
CREATE TABLE taxidx
(
  taxtype    int  REFERENCES taxtypes(id),
  tupleid    int  REFERENCES taxdata(tupleid),
  sim        decimal(5,2)
);
CREATE INDEX taxidxidx ON taxidx(sim);
```

In der Indextabelle *taxidx* werden zu jedem Taxonomietyp alle Datensätze mit einer Ähnlichkeit größer null und dem konkreten Ähnlichkeitswert abgelegt. Diese Daten werden beim Eintragen der Datensätze in *taxdata* automatisch erzeugt und müssen natürlich bei Änderungs- oder Löschoperationen gewartet werden.

B.1.2 Suchanfragen

Die direkte Suchanfrage auf den Datensätzen in *taxdata* findet mit einem Verbund über *taxsims* statt. Gesucht werden alle Datensätze, die eine gewünschte Mindestähnlichkeit zu einem vorgegebenen Taxonomietyp aufweisen.

Die folgende Syntax beschreibt eine direkte Anfrage, die alle Datensätze zurückliefert, die zum Taxonomietyp *3* eine Mindestähnlichkeit von *0,9* haben. Die Ergebnisdatensätze werden nach dem Ähnlichkeitswert sortiert ausgegeben.

Syntax 7 Direkte Anfrage

```
SELECT d.tupleid, s.sim
FROM   taxdata d, taxsims s
WHERE  d.tupletype=s.id1
  AND  s.id2 = 3
  AND  s.sim > 0.9
ORDER BY s.sim DESC
```

Die Suchanfrage über die Indextabelle erfolgt auf der Tabelle *taxidx*. Dazu muß zur Bestimmung des Ergebnisses lediglich der Taxonomietyp und die gewünschte Mindestähnlichkeit angegeben werden.

Syntax 8 Anfrage über Indextabelle

```
SELECT tupleid, sim
FROM   taxidx
WHERE  taxtype = 3
  AND  sim > 0.9
ORDER BY sim DESC
```

B.1.3 Meßergebnisse

Es werden hintereinander zehn Anfragen durchgeführt. Bei jeder Anfrage wird der zu suchende Taxonomiebegriff verändert. Die Anfrage wird jeweils mit einem Direktzugriff und einem Indexzugriff abgearbeitet.

25.000 Tupel. Die erste Messung erfolgt mit 25.000 Tupeln in *taxdata*. Entsprechend der dort vorhandenen Datensätze wurden 40.074 Tupel in der Indextabelle *taxidx* erzeugt.

	Direkte Anfrage	**Anfrage über Index**
Anfrage 1	1,29 sec	1,46 sec
Anfrage 2	2,14 sec	1,93 sec
Anfrage 3	2,01 sec	2,20 sec
Anfrage 4	1,13 sec	0,99 sec
Anfrage 5	1,24 sec	1,69 sec
Anfrage 6	1,23 sec	1,35 sec
Anfrage 7	1,17 sec	1,05 sec
Anfrage 8	1,10 sec	0,97 sec
Anfrage 9	1,16 sec	1,17 sec
Anfrage 10	1,21 sec	1,26 sec
Durchschnitt	**1,39 sec**	**1,31 sec**

50.000 Tupel. Die zweite Messung erfolgt mit 50.000 Tupeln in *taxdata*. Dazu wurden 80.254 Tupel in *taxidx* erzeugt.

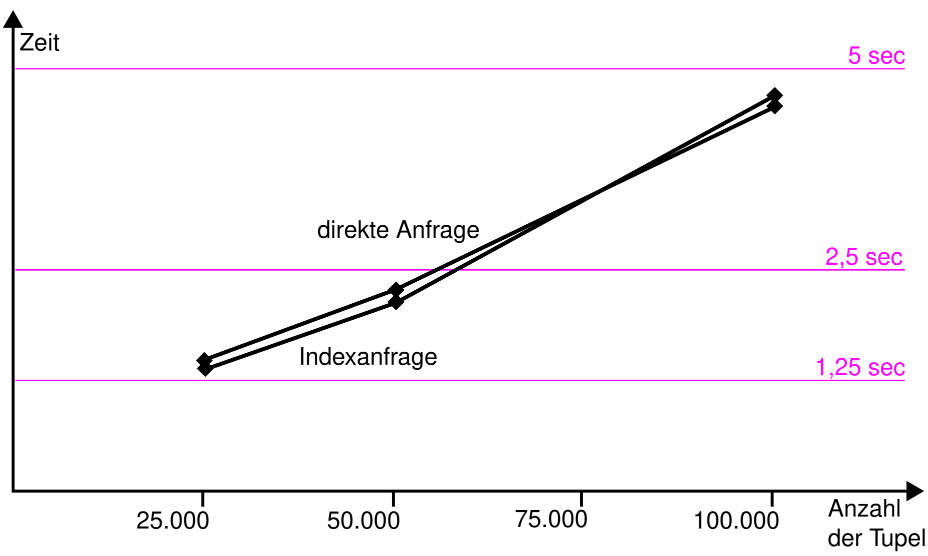
	Direkte Anfrage	**Anfrage über Index**
Anfrage 1	1,92 sec	2,62 sec
Anfrage 2	3,19 sec	3,41 sec
Anfrage 3	3,41 sec	3,43 sec
Anfrage 4	2,04 sec	1,84 sec
Anfrage 5	2,77 sec	1,84 sec
Anfrage 6	1,88 sec	1,72 sec
Anfrage 7	1,97 sec	1,73 sec
Anfrage 8	2,00 sec	1,77 sec
Anfrage 9	1,86 sec	1,70 sec
Anfrage 10	1,92 sec	1,78 sec
Durchschnitt	**2,30 sec**	**2,18 sec**

100.000 Tupel. Die letzte Messung erfolgt mit 100.000 Tupeln in *taxdata*. Dazu wurden 160.268 Tupel in *taxidx* erzeugt.

	Direkte Anfrage	Anfrage über Index
Anfrage 1	3,93 sec	5,47 sec
Anfrage 2	6,75 sec	9,29 sec
Anfrage 3	8,79 sec	6,82 sec
Anfrage 4	3,77 sec	3,36 sec
Anfrage 5	3,36 sec	3,41 sec
Anfrage 6	3,83 sec	3,60 sec
Anfrage 7	3,52 sec	3,41 sec
Anfrage 8	3,76 sec	3,84 sec
Anfrage 9	3,45 sec	3,45 sec
Anfrage 10	3,50 sec	3,39 sec
Durchschnitt	**4,46 sec**	**4,60 sec**

Die folgende Abbildung verdeutlicht noch einmal die Ausführungszeiten der Suchanfragen in Abhängigkeit der Tupelanzahl in der Datentabelle.

Abbildung 3 Ausführungszeiten für taxonomische Daten



Die Anfrage über die Indextabelle benötigt etwa die gleiche Zeit wie die direkte Anfrage auf der Datentabelle. Dies ist darauf zurückzuführen, daß die Ähnlichkeitswerte zwischen zwei Datensätzen nicht mit einer Funktion berechnet werden müssen, sondern direkt über einen Verbund bestimmt werden können. Somit kann das Datenbanksystem alle Möglichkeiten der Anfrageoptimierung einsetzen.

B.2 Messungen mit lokalem Index für ganzzahlige Daten

In diesem Abschnitt führen wir die Leistungsmessungen für die in Kapitel 5 im Abschnitt 5.2.3 eingeführten lokalen Indizes für ganzzahlige Daten durch.

B.2.1 Datenbankschema

Zur Leistungsmessung der Ähnlichkeitsberechnung von ganzzahligen Daten wird zuerst die Tabelle *intdata* angelegt. In dieser Tabelle werden die Datensätze mit dem ganzzahligen Attribut *data* abgelegt.

Syntax 9 Tabelle intdata mit ganzzahligen Daten

```
CREATE TABLE intdata
(
  id      int PRIMARY KEY,
  data    int NOT NULL
);
```

Zur Berechnung eines Ähnlichkeitswertes zwischen zwei ganzzahligen Werten wird eine lineare Ähnlichkeitsfunktion *simInt()* definiert. Für jede Abweichung der beiden Zahlenwerte um *1* nimmt die Ähnlichkeit um *0,15* ab. Die Funktion ist in der *Informix* eigenen Programmiersprache *SPL* implementiert und wird in der folgenden Syntax dargestellt.

Syntax 10 Ähnlichkeitsfunktion simInt

```
CREATE FUNCTION simInt (datavalue int, searchvalue int) RETURNS decimal(5,2)
  DEFINE sim decimal (10,2);
  LET sim = 1 - 0.15 * ABS(searchvalue - datavalue);
  IF (sim < 0) THEN LET sim = 0; END IF;
  RETURN sim;
END FUNCTION;
```

Auf eine Implementierung der Ähnlichkeitsfunktion *simInt()* in Java wurde hier zur Messung verzichtet, da sich die Laufzeit (in mehreren Versuchen geprüft) um etwa den Faktor vier erhöht hat.

Die Datentabelle *intdata* wird mit einer benutzerdefinierten Funktion gefüllt, die zur Erzeugung der Daten einen Zufallsgenerator benutzt. Somit werden zur Messung zwischen 10.000 und 100.000 Datensätze generiert.

Zur Unterstützung der Suche mit einem Index wird die Tabelle *intidx* angelegt. Darin wird notiert, zu welchem Suchwert *value* ein Datensatz *tupleid* welche Ähnlichkeit *sim* besitzt.

Syntax 11 Indextabelle intidx

```
CREATE TABLE intidx
(
  value   int,
  tupleid int  REFERENCES intdata(id),
  sim     decimal(5,2)
);
CREATE INDEX intidxidx ON intidx(value);
```

Nach dem Einfügen eines Datensatzes in *intdata* wird der Wert des Attributes *data* erhöht bzw. erniedrigt, solange die berechnete Ähnlichkeit zu dem ursprünglichen Wert größer null ist. Die dabei errechneten Ähnlichkeitswerte werden in der Indextabelle *intidx* zusammen mit einer Referenz zum Datensatz abgelegt.

Zur Unterstützung des Zugriffs auf die Tabelle *intidx* wird bei der Definition ein Index auf dem Attribut *value* angelegt.

B.2.2 Suchanfragen

Die direkte Suchanfrage auf den Datensätzen der Tabelle *intdata* findet mit einem Aufruf der Funktion *simInt()* statt, die für jede Instanz eine Ähnlichkeit zum Suchwert und damit dessen Qualifikation für die Ergebnismenge berechnet.

Die folgende Syntax zeigt ein Beispiel für die Suche nach Datensätzen, deren Ähnlichkeit des Attributs *data* zum Wert *1000* größer *0.7* sein soll.

Syntax 12 Direkte Suchanfrage

```
SELECT id,data,simint(data,1000) AS sim
FROM   intdata
WHERE  simint(data,1000) > 0.7
ORDER BY sim DESC
```

Der zweimalige Aufruf der Funktion *simInt()* spielt für die Messung der Auswertungszeit keine Rolle, da der Optimierer des Datenbanksystems die beiden Aufrufe miteinander identifiziert und nur einmal ausführt.

Bei der Anfrage über die Indextabelle *intidx* ist ein Verbund mit der Tabelle *intdata* nötig, der zu einem Datensatz in der Ergebnismenge den konkreten Wert des Attributs *data* liefert.

Syntax 13 Suchanfrage über Index

```
SELECT i.tupleid, d.data, i.sim
FROM   intidx i, intdata d
WHERE  i.tupleid = d.id
  AND  i.value = 1000
  AND  i.sim > 0.7
ORDER BY i.sim DESC
```

B.2.3 Meßergebisse

Es werden hintereinander zehn Anfragen durchgeführt. Bei jeder Anfrage wird der zu suchende Wert verändert. Die Anfrage wird jeweils mit einem Direktzugriff und einem Indexzugriff abgearbeitet.

10.000 Tupel. Die erste Messung erfolgt mit 10.000 Tupeln in der Tabelle *intdata*. Zu den Datensätzen in *intdata* wurden 130.000 Tupel in der Tabelle *intidx* erzeugt.

	Direkte Anfrage	**Anfrage über Index**
Anfrage 1	2,52 sec	0,50 sec
Anfrage 2	2,23 sec	0,49 sec
Anfrage 3	2,22 sec	0,47 sec
Anfrage 4	2,22 sec	0,50 sec
Anfrage 5	2,25 sec	0,75 sec
Anfrage 6	2,23 sec	0,71 sec
Anfrage 7	2,50 sec	0,47 sec
Anfrage 8	2,21 sec	0,43 sec
Anfrage 9	2,26 sec	0,46 sec
Anfrage 10	2,24 sec	0,47 sec
Durchschnitt	**2,28 sec**	**0,53 sec**

25.000 Tupel. Die zweite Messung erfolgt mit 25.000 Tupeln in der Tabelle *intdata*. Zu den Datensätzen in *intdata* wurden 325.000 Tupel in der Tabelle *intidx* erzeugt.

	Direkte Anfrage	**Anfrage über Index**
Anfrage 1	5,12 sec	0,44 sec
Anfrage 2	4,96 sec	0,71 sec
Anfrage 3	4,99 sec	0,44 sec
Anfrage 4	5,00 sec	0,42 sec
Anfrage 5	5,03 sec	0,45 sec
Anfrage 6	4,97 sec	0,41 sec
Anfrage 7	4,97 sec	0,44 sec
Anfrage 8	4,96 sec	0,43 sec
Anfrage 9	4,97 sec	0,42 sec
Anfrage 10	4,94 sec	0,41 sec
Durchschnitt	**4,99 sec**	**0,46 sec**

50.000 Tupel. Die dritte Messung erfolgt mit 50.000 Tupeln in der Tabelle *intdata*. Zu den Datensätzen in *intdata* wurden 650.000 Tupel in der Tabelle *intidx* erzeugt.

	Direkte Anfrage	**Anfrage über Index**
Anfrage 1	9,72 sec	0,61 sec
Anfrage 2	9,59 sec	0,60 sec
Anfrage 3	9,66 sec	0,52 sec
Anfrage 4	9,56 sec	0,56 sec
Anfrage 5	9,57 sec	0,56 sec
Anfrage 6	9,55 sec	0,66 sec
Durchschnitt	**9,57 sec**	**0,59 sec**

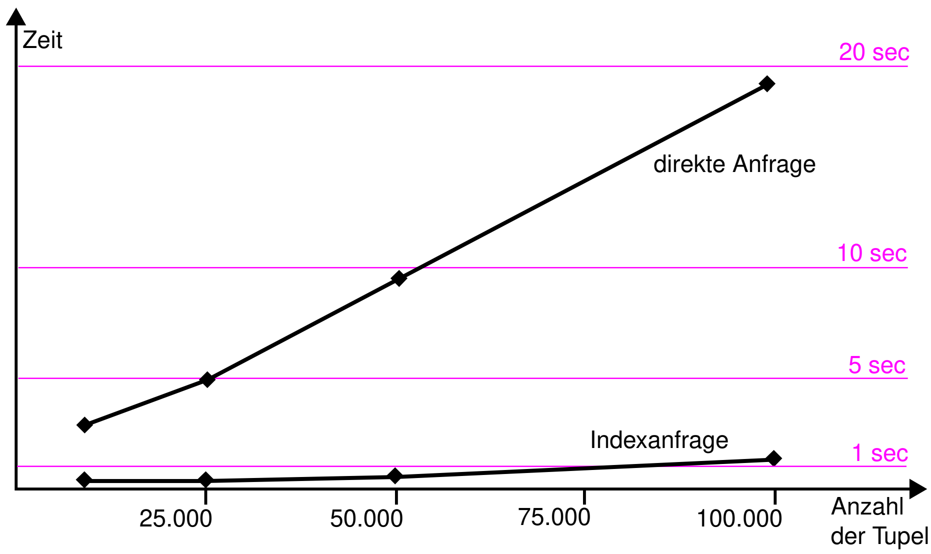
	Direkte Anfrage	Anfrage über Index
Anfrage 7	9,55 sec	0,57 sec
Anfrage 8	9,54 sec	0,58 sec
Anfrage 9	9,52 sec	0,58 sec
Anfrage 10	9,52 sec	0,66 sec
Durchschnitt	**9,57 sec**	**0,59 sec**

100.000 Tupel. Die letzte Messung erfolgt mit 100.000 Tupeln in der Tabelle *intdata*. Zu den Datensätzen in *intdata* wurden 1.300.000 Tupel in der Tabelle *intidx* erzeugt.

	Direkte Anfrage	Anfrage über Index
Anfrage 1	18,87 sec	1,47 sec
Anfrage 2	18,63 sec	1,68 sec
Anfrage 3	18,65 sec	1,35 sec
Anfrage 4	18,63 sec	1,25 sec
Anfrage 5	18,54 sec	0,76 sec
Anfrage 6	19,32 sec	0,97 sec
Anfrage 7	19,05 sec	0,90 sec
Anfrage 8	18,99 sec	1,01 sec
Anfrage 9	19,00 sec	1,02 sec
Anfrage 10	18,99 sec	1,02 sec
Durchschnitt	**18,86 sec**	**1,14 sec**

Die folgende Abbildung verdeutlicht noch einmal die Ausführungszeiten der Suchanfragen in Abhängigkeit der Tupelanzahl in der Datentabelle.

Abbildung 4 Ausführungszeiten für ganzzahlige Daten



Bei einer Datenmenge von 100.000 Tupeln benötigt die Anfrage über die Indextabelle nur noch etwa sechs Prozent der Ausführungszeit der direkten Anfrage auf der Datentabelle.

B.3 Function-value-indexing

Eine weitere Möglichkeit, die Suchanfrage zu unterstützen ist das *function-value-indexing* (Abschnitt 5.2.6). Dabei wird ein Index auf einer Tabelle angelegt, der die zu indexierenden Daten nicht direkt aus einer Tabellenspalte bezieht, sondern von einer Funktion, die ihren Funktionswert aus einer angegebenen Tabellenspalte berechnet.

Der große Nachteil dieses Verfahrens besteht jedoch darin, daß die Funktion, die die zu indexierenden Daten errechnet, als Parameter ausschließlich Spalten der Tabelle, auf der der Index definiert wird, besitzen darf. Das bedeutet, daß der Suchwert fest in den Funktionsrumpf kodiert werden muß und somit für jeden Suchwert ein separater Index angelegt werden müßte.

Ganzzahlige Daten. Die nächste Syntax stellt die Definition einer Ähnlichkeitsfunktion für ganzzahlige Daten mit einem dazugehörigen *Function-value-index* dar. Dazu werden im folgenden die in Abschnitt B.2.1 eingeführten Tabellen *intdata* und *intidx* verwendet.

Syntax 14 Function-value-indexing

```
CREATE FUNCTION simInt2000 (datavalue int) RETURNS decimal(5,2) WITH (NOT VARIANT)
  RETURN simInt (datavalue, 2000);
END FUNCTION;

CREATE INDEX intIdx2000 ON intdata (simInt2000(data));
```

Die Funktion *simInt2000()* berechnet einen Ähnlichkeitswert des übergebenen Parameters *datavalue* zum Wert *2000* mit Hilfe der Funktion *simInt()*. Für die Suche nach dem Wert *2000* wird dann der *Function-value-index intIdx2000* angelegt. Die Suchanfrage hat dann die folgende Gestalt.

Syntax 15 Suchanfrage

```
SELECT id,data,simint2000(data) AS sim
FROM   intdata
WHERE  simint2000(data) > 0.7
ORDER BY sim DESC
```

Messungen haben ergeben, daß eine Suche über die Ähnlichkeitsfunktion *simInt2000()* etwa genau so schnell abgearbeitet wird, wie der Zugriff über die Indextabelle *intidx*. Allerdings ist hierbei nicht berücksichtigt, daß zum Aufbau einer solchen Suchanfrage zunächst erst noch bestimmt werden muß, ob für einen allgemein gesuchten Wert eine spezielle Ähnlichkeitsfunktion mit angelegtem *Function-value-index* vorhanden ist. Nach dieser Überprüfung muß eine Suchanfrage mit einer entsprechenden Ähnlichkeitsfunktion generiert werden, die danach erst ausgeführt werden kann.

Ein weiterer Nachteil ist, daß solch spezielle Ähnlichkeitsfunktionen mit dazugehörenden Indizes nur für besonders häufig auftretende Suchwerte erstellt werden können. Mit Berücksichtigung der Auswahl einer entsprechenden Funktion gemäß dem Suchwert und Generierung der Suchanfrage sind damit keine Leistungsvorteile zu erwarten.

Es wurden fünf Messungen auf der Tabelle *intdata* mit 100.000 Tupeln durchgeführt. Bei jeder Messung wurde eine Anfrage nach einem anderen Suchwert gestartet. Protokolliert wird die Ausführungszeit für einen Zugriff über die Indextabelle *intidx* und die Zeit für einen direkten Zugriff auf die Datentabelle *intdata* mit einem zuvor angelegten *Function-value-index*.

Suchwert	Direkte Anfrage mit Function-value-indexing	Anfrage über Indextabelle
2000	0,47 sec	0,55 sec
3000	0,49 sec	0,52 sec
4000	0,47 sec	0,51 sec
5000	0,42 sec	0,49 sec
6000	0,43 sec	0,42 sec
Durchschnitt	**0,47 sec**	**0,50 sec**

Der Vorteil einer Suchanfrage über die Indextabelle *intidx* besteht gegenüber dem *Function-value-indexing* darin, daß keine Anfrage generiert werden muß und die in Abschnitt B.2.3 und diesem Abschnitt gemessenen Leistungsdaten für eine Anfrage über die Indextabelle für jeden beliebigen Suchwert gelten.

Die abweichenden Anfragezeiten in Abschnitt B.2.3 und diesem Abschnitt bei einem Indexzugriff mit 100.000 Tupeln in der Datentabelle sind dadurch zu erklären, daß die Messungen an verschiedenen Tagen (und damit unterschiedlicher Last) durchgeführt worden sind.

Taxonomische Daten. Wie in Abschnitt B.1.3 erläutert wurde, macht es sich in der Antwortzeit für eine Ähnlichkeitssuche auf Taxonomien nicht bemerkbar, ob ein direkter Zugriff auf der Datentabelle oder ein Zugriff über die Indextabelle durchgeführt wurde. Auch hier stellt sich die Frage, ob der Einsatz der *Function-value-indexing* Technik Leistungsvorteile ermöglicht.

Dazu wird eine Funktion angelegt, die für einen gegebenen Suchwert die Ähnlichkeit zu einem als Parameter übergebenen Attributwert berechnet. Der Ähnlichkeitswert ist dabei fest in den Funktionsrumpf kodiert. In einem realen System könnte der Funktionsrumpf mit den Daten aus der Tabelle *taxsims* (Abschnitt B.1.1) generiert werden.

Syntax 16 Function-value-indexing

```
CREATE FUNCTION simTax3 (datavalue int) RETURNS decimal(5,2) WITH (NOT VARIANT)
  IF   datavalue == 1 THEN RETURN 0.7;
  ELIF datavalue == 2 THEN RETURN 0.95;
  ELIF datavalue == 3 THEN RETURN 1.0;
  ELSE RETURN 0.0;
  END IF;
END FUNCTION;

CREATE INDEX taxidx3 ON taxdata (simTax3(tupletype));
```

Die Funktion *simTax3()* berechnet den Ähnlichkeitswert eines übergebenen Attributwertes zum Suchwert *3*. Nach dem Anlegen der Funktion wird ein *Function-value-index* auf der Tabelle t*axdata* definiert.

Nun werden fünf Messungen bei 100.000 Tupeln in der Tabelle *taxdata* mit verschiedenen Suchwerten durchgeführt. Dabei erfolgt ein Zugriff direkt auf der Datentabelle mit Verbund über die Tabelle *taxsims*, die die Ähnlichkeitswerte verwaltet. Ein weiterer Zugriff wird mit einer Ähnlichkeitsfunktion und angelegtem *Function-value-index* durchgeführt, dabei ist die Funktion und der Index für jeden Suchwert neu zu definieren.

Suchwert	Direkte Anfrage mit Function-value-indexing	Direkte Anfrage mit Verbund
1	5,74 sec	3,55 sec
2	11,12 sec	6,44 sec
3	10,78 sec	6,43 sec
4	5,84 sec	3,54 sec
5	5,45 sec	3,31 sec
Durchschnitt	**7,79 sec**	**4,66 sec**

Eine direkte Anfrage auf die Datentabelle mit Einsatz eines *Function-value-index* bringt keine Leistungsvorteile gegenüber einer Anfrage mit Verbund über die Ähnlichkeitstabelle *taxsims*. Die Antwortzeit mit *Function-value-index* ist deutlich höher. Zusätzlich muß für jeden Suchwert eine Funktion und ein Index angelegt werden, bei einer konkreten Anfrage zunächst die Funktion für den entsprechenden Suchwert bestimmt und die Suchanfrage mit Verwendung der ermittelten Funktion generiert werden.

B.4 Messungen mit lokalem Index für Fließkommadaten

In diesem Abschnitt führen wir die Leistungsmessungen für die in Kapitel 5 im Abschnitt 5.2.4 eingeführten lokalen Indizes für Fließkommadaten durch.

B.4.1 Datenbankschema

Zur Leistungsmessung der Ähnlichkeitsberechnung von Fließkommadaten wird zuerst die Tabelle *decdata* angelegt. In dieser Tabelle werden die Datensätze mit dem Fließkommaattribut *data* abgelegt.

Syntax 17 Tabelle decdata mit Fließkommadaten

```
CREATE TABLE decdata
(
  id      int PRIMARY KEY,
  data    decimal(3,2) NOT NULL
);
```

Zur Berechnung eines Ähnlichkeitswertes zwischen zwei Fließkommawerten wird eine lineare Ähnlichkeitsfunktion *simDec()* definiert. Für jede Abweichung der beiden Zahlenwerte um *0,01* nimmt die Ähnlichkeit um *0,08* ab. Die Funktion ist in der *Informix* eigenen Programmiersprache *SPL* implementiert und wird in der folgenden Syntax dargestellt.

Syntax 18 Ähnlichkeitsfunktion simDec

```
CREATE FUNCTION simDec (datavalue decimal(3,2), searchvalue decimal(3,2))
      RETURNS decimal(5,2)
  DEFINE sim decimal (10,2);
  DEFINE datavalueInt   int;
  DEFINE searchvalueInt int;
  LET datavalueInt   = (datavalue   * 100)::int;
  LET searchvalueInt = (searchvalue * 100)::int;
  LET sim = 1 - 0.08 * ABS(searchvalueInt - datavalueInt);
  IF (sim < 0) THEN LET sim = 0; END IF;
  RETURN sim;
END FUNCTION;
```

Die Datentabelle *decdata* wird mit einer benutzerdefinierten Funktion gefüllt, die zur Erzeugung der Daten einen Zufallsgenerator benutzt. Somit werden zur Messung zwischen 10.000 und 100.000 Datensätze generiert.

Zur Unterstützung der Suche mit einem Index wird die Tabelle *decidx* angelegt. Darin wird notiert, zu welchem Suchwert *value* ein Datensatz *tupleid* welche Ähnlichkeit *sim* besitzt. Dabei ist *value* ein ganzzahliger Wert, der durch Multiplikation des Attributwertes des Datensatzes mit dem Faktor *100* und anschließender Rundung entsteht.

Syntax 19 Indextabelle decidx

```
CREATE TABLE decidx
(
  value   int,
  tupleid int  REFERENCES decdata(id),
  sim     decimal(5,2)
);
CREATE INDEX decidxidx ON decidx(value);
```

Nach dem Einfügen eines Datensatzes in *decdata* wird der Wert des Attributes *data* mit *100* multipliziert und um den Wert *2* erhöht bzw. erniedrigt, solange die berechnete Ähnlichkeit zu dem ursprünglichen Wert größer null ist. Dadurch halbieren wir in etwa die Menge der erzeugten Indexdaten. Die errechneten Ähnlichkeitswerte werden in der Indextabelle *decidx* zusammen mit einer Referenz zum Datensatz abgelegt.

Zur Unterstützung des Zugriffs auf die Tabelle *decidx* wird bei der Definition ein Index auf dem Attribut *value* angelegt.

B.4.2 Suchanfragen

Die direkte Suchanfrage auf den Datensätzen der Tabelle *decdata* findet mit einem Aufruf der Funktion *simDec()* statt, die für jede Instanz eine Ähnlichkeit zum Suchwert und damit dessen Qualifikation für die Ergebnismenge berechnet.

Die folgende Syntax zeigt ein Beispiel für die Suche nach Datensätzen, deren Ähnlichkeit des Attributs *data* zum Wert *0,86* größer *0,9* sein soll.

Syntax 20 Direkte Suchanfrage

```
SELECT id, data, simdec (data, 0.86) AS sim
FROM   decdata
WHERE  simdec (data, 0.86) > 0.90
ORDER BY 3 DESC
```

Der zweimalige Aufruf der Funktion *simDec()* spielt für die Messung der Auswertungszeit keine Rolle, da der Optimierer des Datenbanksystems die beiden Aufrufe miteinander identifiziert und nur einmal ausführt.

Bei der Anfrage über die Indextabelle *decidx* ist ein Verbund mit der Tabelle *decdata* nötig, der zu einem Datensatz in der Ergebnismenge den konkreten Wert des Attributs *data* liefert. Dieser ist ebenfalls zur exakten Berechnung des Ähnlichkeitswertes mit der Ähnlichkeitsfunktion *simDec()* erforderlich.

Syntax 21 Suchanfrage über Index

```
SELECT tupleid, data, simdec(data,0.86) AS sim
FROM   decidx i, decdata d
WHERE  i.value = 86
  AND  i.sim > 0.90
  AND  i.tupleid = d.id
  AND  simdec(data,0.86) > 0.90
ORDER BY 3 DESC
```

B.4.3 Meßergebisse

Es werden hintereinander fünf Anfragen durchgeführt. Bei jeder Anfrage wird der zu suchende Wert verändert. Die Anfrage wird jeweils mit einem Direktzugriff und einem Indexzugriff abgearbeitet.

10.000 Tupel. Die erste Messung erfolgt mit 10.000 Tupeln in der Tabelle *decdata*. Zu den Datensätzen in *decdata* wurden 124.818 Tupel in der Tabelle *decidx* erzeugt.

	Direkte Anfrage	**Anfrage über Index**
Anfrage 1	3,02 sec	0,80 sec
Anfrage 2	3,12 sec	0,83 sec
Anfrage 3	3,20 sec	0,83 sec
Anfrage 4	3,08 sec	0,81 sec
Anfrage 5	3,08 sec	0,79 sec
Durchschnitt	**3,10 sec**	**0,81 sec**

25.000 Tupel. Die zweite Messung erfolgt mit 25.000 Tupeln in der Tabelle *decdata*. Zu den Datensätzen in *decdata* wurden 311.912 Tupel in der Tabelle *decidx* erzeugt.

	Direkte Anfrage	Anfrage über Index
Anfrage 1	7,34 sec	1,63 sec
Anfrage 2	7,87 sec	1,50 sec
Anfrage 3	7,90 sec	1,55 sec
Anfrage 4	8,20 sec	1,55 sec
Anfrage 5	7,88 sec	1,55 sec
Durchschnitt	**7,84 sec**	**1,56 sec**

50.000 Tupel. Die dritte Messung erfolgt mit 50.000 Tupeln in der Tabelle *decdata*. Zu den Datensätzen in *decdata* wurden 624.039 Tupel in der Tabelle *decidx* erzeugt.

	Direkte Anfrage	Anfrage über Index
Anfrage 1	13,97 sec	4,18 sec
Anfrage 2	14,30 sec	4,08 sec
Anfrage 3	14,37 sec	4,05 sec
Anfrage 4	15,03 sec	4,08 sec
Anfrage 5	15,24 sec	4,03 sec
Durchschnitt	**14,58 sec**	**4,08 sec**

75.000 Tupel. Die vierte Messung erfolgt mit 75.000 Tupeln in der Tabelle *decdata*. Zu den Datensätzen in *decdata* wurden 935.699 Tupel in der Tabelle *decidx* erzeugt.

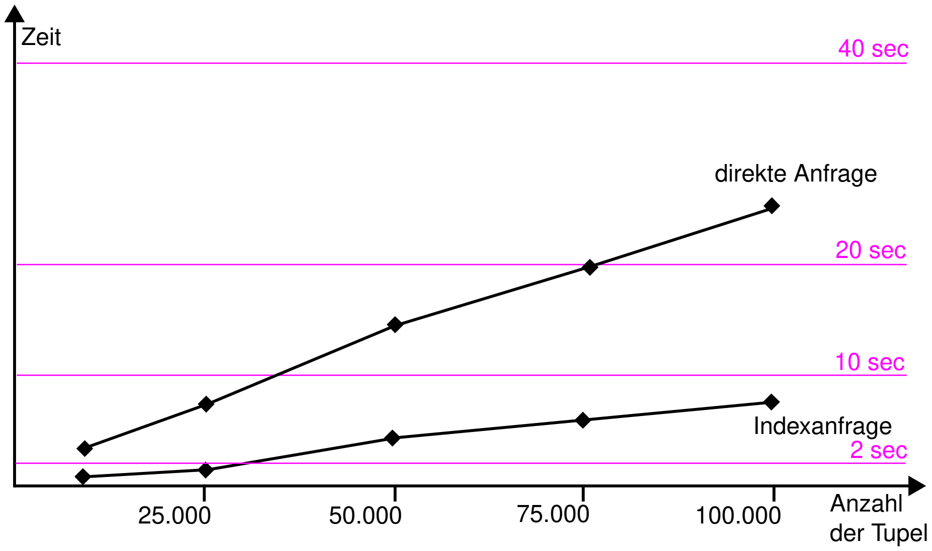
	Direkte Anfrage	Anfrage über Index
Anfrage 1	20,71 sec	5,78 sec
Anfrage 2	19,62 sec	5,68 sec
Anfrage 3	19,08 sec	5,94 sec
Anfrage 4	18,99 sec	5,91 sec
Anfrage 5	18,74 sec	5,94 sec
Durchschnitt	**19,43 sec**	**5,85 sec**

100.000 Tupel. Die letzte Messung erfolgt mit 100.000 Tupeln in der Tabelle *decdata*. Zu den Datensätzen in *decdata* wurden 1.247.456 Tupel in der Tabelle *decidx* erzeugt.

	Direkte Anfrage	Anfrage über Index
Anfrage 1	25,15 sec	7,59 sec
Anfrage 2	25,42 sec	7,75 sec
Anfrage 3	25,40 sec	7,88 sec
Anfrage 4	25,22 sec	7,72 sec
Anfrage 5	25,31 sec	7,77 sec
Durchschnitt	**25,30 sec**	**7,74 sec**

Die folgende Abbildung verdeutlicht noch einmal die Ausführungszeiten der Suchanfragen in Abhängigkeit der Tupelanzahl in der Datentabelle.

Abbildung 5 Ausführungszeiten für Fließkommadaten



Ab einer Datenmenge von 50.000 Tupeln benötigt die Anfrage über die Indextabelle noch etwa 30 Prozent der Ausführungszeit der direkten Anfrage auf der Datentabelle.

B.5 Messungen mit lokalem Index für Zeichenketten

In diesem Abschnitt führen wir die Leistungsmessungen für die in Kapitel 5 im Abschnitt 5.2.5 eingeführten lokalen Indizes für Zeichenketten durch.

B.5.1 Datenbankschema

Zur Leistungsmessung der Ähnlichkeitsberechnung von Zeichenketten (Strings) wird zunächst die Tabelle *ngramdata* angelegt, die die Datensätze, bestehend aus einer ID und einer Zeichenkette bis maximal 20 Zeichen, verwaltet.

Syntax 22 Datentabelle ngramdata

```
CREATE TABLE ngramdata
(
  id      int PRIMARY KEY,
  name    char(20)
);
```

Zur Berechnung des Ähnlichkeitswertes zweier Strings wird die benutzerdefinierte Ähnlichkeitsfunktion *simTrigram()* angelegt, die in Java implementiert ist.

Syntax 23 Ähnlichkeitsfunktion simTrigram

```
public static String simtrigram (String value, String trigramList) throws
                                                                SQLException
{
    int       commonTrigrams = 0;  // number of common trigrams in parameters
    int       maxNumber      = 0;  // max number of trigrams    in parameters
    Vector    trigrams       = new Vector();   // store trigramList in vector

    value       = value.trim();
    trigramList = trigramList.trim();

    for (int i = 0; i < value.length()-2; i++)
    {
        trigrams.addElement (value.substring(i,i+3).toLowerCase());
    }
    StringTokenizer strTok  = new StringTokenizer (trigramList, ",");
    int             number  = strTok.countTokens();
    String          trigram = "";  // current trigram while tokenizer loop
    for (int i = 0; i < number; i++)
    {
        trigram = strTok.nextToken();
        if (trigrams.contains (trigram)) commonTrigrams++;
    }

    if (commonTrigrams == 0) return "0";
    if (trigrams.size() > number) maxNumber = trigrams.size();
    else maxNumber = number;

    return "" + ((double)commonTrigrams / (double)maxNumber);
}
```

SimTrigram() erwartet als Parameter ein Wort *value* und den Suchbegriff als Liste von Trigrams. Die Ähnlichkeit des Wortes zu dem Suchbegriffs wird als Zahlenwert zwischen *0* und *1* zurückgeliefert. Der Suchbegriff wird als eine Aufzählung seiner Trigrams übergeben, da diese Zerlegung bei jeder Berechnung identisch durchgeführt werden muß und somit auch schon als Optimierung beim direkten Zugriff auf die Datentabelle *ngramdata* genutzt werden kann.

In der Funktion wird der String *value* zunächst in Trigrams zerlegt und diese im Java-Vector *trigrams* gespeichert. Danach werden die Trigrams aus der Liste *trigramList* bestimmt und überprüft, ob diese bereits im Vector *trigrams* vorhanden sind.

Um die Ähnlichkeitssuche auf *ngramdata* zu unterstützen, wird die Indextabelle *ngramidx* angelegt, in der zu einem Trigram die Referenz zu einem Datensatz aus *ngramdata* abgelegt wird, der dieses Trigram enthält.

Syntax 24 Indextabelle ngramidx

```
CREATE TABLE ngramidx
(
  trigram   char(3),
  dataid    int      REFERENCES ngramdata(id)
);
CREATE INDEX ngramidxidx ON ngramidx (trigram);
```

Beim Einfügen eines Datensatzes in die Tabelle *ngramdata* wird die Zeichenkette *name* in ihre Trigrams zerlegt und diese zusammen mit der entsprechenden Referenz in der Tabelle *ngramidx* gespeichert. Die Indexeinträge müssen bei Änderungs- und Löschoperationen auf der Tabelle *ngramdata* gewartet werden.

Zur Unterstützung des Zugriffs auf die Indextabelle *ngramidx* wird ein Index auf dem Attribut *trigram* angelegt.

Um einen Ähnlichkeitswert mit der Indextabelle *ngramidx* berechnen zu können, benötigt man zusätzlich noch die Anzahl der Trigrams. Diese werden von der benutzerdefinierten Funktion *countTrigrams()* berechnet.

Syntax 25 Berechnung der Anzahl der Trigrams

```
CREATE FUNCTION countTrigrams (name char(20), no int, out number int) RETURNS int
  LET number = length(trim(name)) - 2; -- length-2 == number of trigrams
  IF (number < no) THEN LET number = no; END IF; -- return max of number and no
  RETURN number;
END FUNCTION;
```

Die Datentabelle *ngramdata* wird mit einer benutzerdefinierten Funktion gefüllt. Diese erzeugt Zeichenketten, die aus einer nach dem Zufallsgenerator bestimmten Folge von Buchstaben bestehen. Die Länge der Zeichenketten wird ebenfalls mit dem Zufallsgenerator zwischen acht und dreizehn Zeichen gewählt.

B.5.2 Suchanfragen

Die direkte Suchanfrage auf den Datensätzen erfolgt über die Tabelle *ngramdata* mit Verwendung der Ähnlichkeitsfunktion *simTrigram()*.

Die folgende Syntax beschreibt die Suche nach Datensätzen, deren Ähnlichkeitswert des Attributs *name* zum Wort *eintrag* größer *0,2* ist. Das Suchwort wird schon als Aufzählung von Trigrams übergeben.

Syntax 26 Direkte Suchanfrage

```
SELECT id, name, simTrigram (name, 'ein,int,ntr,tra,rag') AS sim
FROM   ngramdata
WHERE  simTrigram (name, 'ein,int,ntr,tra,rag') > 0.2
ORDER BY sim DESC
```

Der doppelte Aufruf der Funktion *simTrigram()* spielt für die Messung der Ausführungszeit keine Rolle, da der Anfrageoptimierer die beiden Aufrufe miteinander identifiziert und nur einmal ausführt.

Die Suchanfrage über die Indextabelle benötigt einen Verbund mit der Datentabelle, da zu jedem gefundenen Ergebnistupel auch der Wert des Attributs *name* ausgegeben werden soll.

Syntax 27 Suchanfrage über Indexstruktur

```
SELECT dataid, name, number/totalnumber AS sim
FROM
TABLE((MULTISET(SELECT dataid, name, COUNT(dataid) AS number, totalnumber
                FROM   ngramidx, ngramdata
                WHERE  (trigram='ein' OR trigram='int' OR trigram='ntr' OR
                        trigram='tra' OR trigram='rag')
                  AND countTrigrams (name,5,totalnumber # int) > 0
                  AND id=dataid
                GROUP BY 1,2,4)))
WHERE number/totalnumber > 0.2
ORDER BY 3 DESC
```

Zunächst werden in der inneren *SELECT* Klausel alle Datensätze bestimmt, die zumindest ein Trigram des Suchbegriffs enthalten, und die Datensatz ID zusammen mit der Anzahl der übereinstimmenden Trigrams an die äußere *SELECT* Klausel übergeben. Diese berechnet dann mit den erhaltenen Daten einen Ähnlichkeitswert.

B.5.3 Meßergebnisse

Es werden hintereinander fünf Suchanfragen durchgeführt. Bei jeder Suchanfrage wird der zu suchende Begriff verändert. Die Anfrage wird jeweils mit einem Direktzugriff und einem Indexzugriff abgearbeitet.

10.000 Tupel. Die erste Messung erfolgt mit 10.000 Tupeln in *ngramdata*. Entsprechend der dort vorhandenen Datensätze wurden 83.928 Tupel in der Indextabelle *ngramidx* erzeugt.

	Direkte Anfrage	**Anfrage über Index**
Anfrage 1	9,42 sec	0,54 sec
Anfrage 2	9,15 sec	0,50 sec
Anfrage 3	9,36 sec	0,50 sec
Anfrage 4	9,16 sec	0,50 sec
Anfrage 5	9,12 sec	0,49 sec
Durchschnitt	**9,24 sec**	**0,51 sec**

25.000 Tupel. Die zweite Messung erfolgt mit 25.000 Tupeln in *ngramdata*. Entsprechend der dort vorhandenen Datensätze wurden 209.516 Tupel in der Indextabelle *ngramidx* erzeugt.

	Direkte Anfrage	**Anfrage über Index**
Anfrage 1	22,83 sec	0,65 sec
Anfrage 2	22,89 sec	0,64 sec
Anfrage 3	22,24 sec	0,63 sec
Anfrage 4	21,72 sec	0,65 sec
Anfrage 5	22,03 sec	0,67 sec
Durchschnitt	**22,34 sec**	**0,65 sec**

50.000 Tupel. Die dritte Messung erfolgt mit 50.000 Tupeln in *ngramdata*. Entsprechend der dort vorhandenen Datensätze wurden 418.496 Tupel in der Indextabelle *ngramidx* erzeugt.

	Direkte Anfrage	Anfrage über Index
Anfrage 1	43,48 sec	0,96 sec
Anfrage 2	43,61 sec	0,95 sec
Anfrage 3	43,26 sec	0,73 sec
Anfrage 4	43,30 sec	0,97 sec
Anfrage 5	44,57 sec	0,90 sec
Durchschnitt	**43,64 sec**	**0,90 sec**

75.000 Tupel. Die vierte Messung erfolgt mit 75.000 Tupeln in *ngramdata*. Entsprechend der dort vorhandenen Datensätze wurden 627.831 Tupel in der Indextabelle *ngramidx* erzeugt.

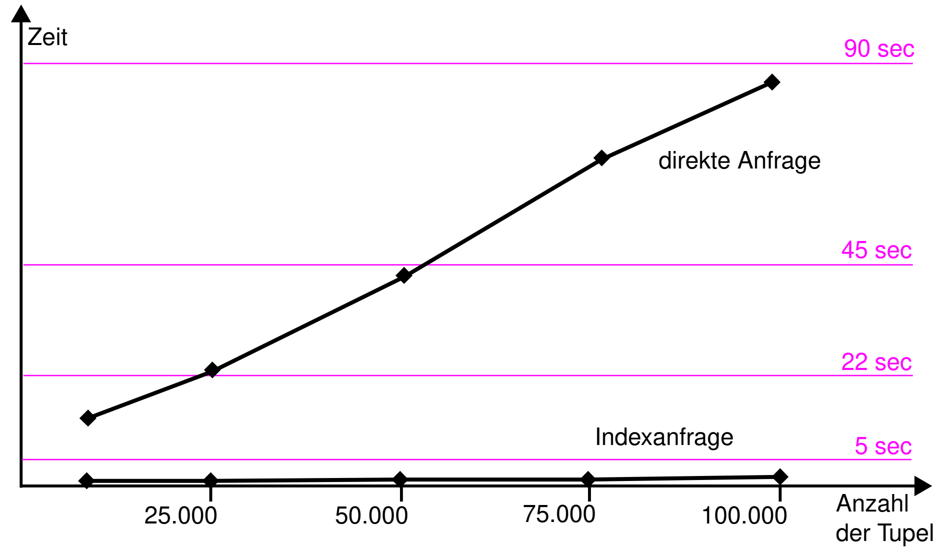
	Direkte Anfrage	Anfrage über Index
Anfrage 1	1:05,82 min	1,29 sec
Anfrage 2	1:05,79 min	1,35 sec
Anfrage 3	1:05,34 min	1,27 sec
Anfrage 4	1:04,83 min	1,28 sec
Anfrage 5	1:05,29 min	1,31 sec
Durchschnitt	**1:05,41 min**	**1,30 sec**

100.000 Tupel. Die letzte Messung erfolgt mit 100.000 Tupeln in *ngramdata*. Entsprechend der dort vorhandenen Datensätze wurden 837.030 Tupel in der Indextabelle *ngramidx* erzeugt.

	Direkte Anfrage	Anfrage über Index
Anfrage 1	1:26,75 min	1,48 sec
Anfrage 2	1:27,82 min	1,57 sec
Anfrage 3	1:25,65 min	1,32 sec
Anfrage 4	1:27,10 min	1,43 sec
Anfrage 5	1:26,67 min	1,24 sec
Durchschnitt	**1:26,80 min**	**1,41 sec**

Die folgende Abbildung verdeutlicht noch einmal die Ausführungszeiten der Suchanfragen in Abhängigkeit der Tupelanzahl in der Datentabelle.

Abbildung 6 Ausführungszeiten für Zeichenketten



Bei einer Datenmenge von 100.000 Tupeln benötigt eine Suchanfrage über die Indextabelle nur noch etwa 1,6 Prozent der Zeit, die zur direkten Suche auf der Datentabelle nötig ist.

B.5.4 Nutzung objekt-relationaler Technologien

Zum Einsatz objekt-relationaler Technologien legen wir die Indexdaten nicht in einer separaten Tabelle ab, sondern speichern die Trigrams einer Zeichenkette in einem zusätzlichen mengenwertigen Attribut.

Syntax 28 Tabelle ngramdata2

```
CREATE TABLE ngramdata2
(
  id        int PRIMARY KEY,
  name      char(20),
  trigrams  LIST (char(3) NOT NULL),
  number    int
);
```

Zusätzlich zum Primärschlüssel *id* und der Zeichenkette *name* des Datensatzes speichern wir die extrahierten Trigrams des Strings im Attribut *trigrams* und die Anzahl der Trigrams im Attribut *number*.

Die Ähnlichkeitssuche erfolgt nun nicht über den Aufruf einer Ähnlichkeitsfunktion, die die Zeichenkette und einen Suchbegriff übergeben bekommt, sondern findet direkt mit einer SQL Anfrage auf den Attributen *trigrams* und *number* statt.

Syntax 29 Anfrage mit objekt-relationaler Technologie

```
SELECT id,name,number/totalnumber AS sim
FROM TABLE((MULTISET(SELECT id,name,(SELECT COUNT(*)
                                     FROM   TABLE((SELECT trigrams
                                                   FROM   ngramdata2
                                                   WHERE  id=d2.id)) t(trigram)
                                     WHERE trigram IN LIST{'ein','int','ntr',
                                                          'tra','rag'}) AS number,
                            CASE WHEN (5 > d2.number) THEN 5 ELSE d2.number END
                            AS totalnumber
                     FROM ngramdata2 d2)))
WHERE number/totalnumber > 0.1
ORDER BY 3 DESC
```

Auch hier werden fünf Messungen mit jeweils verschiedenen Suchbegriffen durchgeführt, um einen Durchschnittswert zu erhalten. Die Messungen werden auf der Datentabelle *ngramdata2* mit 25.000 Tupeln durchgeführt.

	Anfragezeit
Anfrage 1	2:11,60 min
Anfrage 2	2:12,35 min
Anfrage 3	2:12,57 min
Anfrage 4	2:12,45 min
Anfrage 5	2:12,27 min
Durchschnitt	**2:25,25 min**

Da die Zugriffszeit auf das Suchergebnis mit den objekt-relationalen Indexdaten jedoch fast das Sechsfache der direkten Suche auf der Datentabelle mit einer benutzerdefinierten Ähnlichkeitsfunktion beträgt, wurde die Messung nach dem Erstellen obiger Tabelle frühzeitig abgebrochen.

B.6 Messungen mit globalem Index

Um die Leistungsmessungen mit einem globalen Index durchzuführen, legen wir eine Datentabelle mit mehreren Attributen an und erzeugen für jedes Attribut lokale Indexdaten. Basierend auf diesen lokalen Indexdaten wird ein globaler Index für die Datentabelle erzeugt.

Beim Ausführen der Suchanfrage können wir die Antwortzeiten mit Zugriff über den globalen Index, mit Zugriff über einen Verbund der lokalen Indizes und mit Zugriff über Aufruf der Ähnlichkeitsfunktionen für jedes einzelne Attribut miteinander vergleichen.

B.6.1 Datenbankschema

Zunächst legen wir die Datentabelle *abc* mit den drei ganzzahligen Attributen *a*, *b* und *c* an. Für die lokalen Indexdaten der Attribute (analog zu Abschnitt B.2) werden die Indextabellen *aidx*, *bidx* und *cidx* angelegt. Zur Unterstützung des Zugriffs legen wir jeweils noch einen Index an.

Syntax 30 Datentabelle und lokale Indizes

```
CREATE TABLE abc
(
  id    int  PRIMARY KEY,
  a     int  NOT NULL,
  b     int  NOT NULL,
  c     int  NOT NULL
);

CREATE TABLE aidx
(
  value   int,
  tupleid int REFERENCES abc(id),
  sim     decimal(5,2)
);
CREATE INDEX aidxidx ON aidx(value);

CREATE TABLE bidx
(
  value   int,
  tupleid int REFERENCES abc(id),
  sim     decimal(5,2)
);
CREATE INDEX bidxidx ON bidx(value);

CREATE TABLE cidx
(
  value   int,
  tupleid int REFERENCES abc(id),
  sim     decimal(5,2)
);
CREATE INDEX cidxidx ON cidx(value);
```

Zur Berechnung der lokalen Ähnlichkeiten wird ein lineares Ähnlichkeitsmaß gewählt. Für jede Abweichung des Suchwertes um *1* vom Attributwert wird der Ähnlichkeitswert beim Attribut *a* um *0,15* erniedrigt, beim Attribut *b* um *0,2* und beim Attribut *c* um *0,22*. Damit werden für jeden Datensatz in der Datentabelle *abc* insgesamt *13* Einträge in der Tabelle *aidx*, *11* Einträge in der Tabelle *bidx* und *9* Einträge in der Tabelle *cidx* generiert.

Die Daten des globalen Index werden in der Tabelle *abcidx* gespeichert. Auch hier wird zusätzlich ein Index angelegt, der den Zugriff auf die Tabelle beschleunigt.

Syntax 31 Globale Indextabelle

```
CREATE TABLE abcidx
(
  avalue  int,
  bvalue  int,
  cvalue  int,
  tupleid int REFERENCES abc(id),
  sim     decimal(5,2)
);
CREATE INDEX abcidxidx ON abcidx(avalue,bvalue,cvalue);
```

Nachdem ein Datensatz in die Tabelle *abc* eingefügt wurde, werden für diesen Satz lokale Indexdaten in den Tabellen *aidx*, *bidx* und *cidx* (wie oben beschrieben) erzeugt. Für die so entstandenen lokalen Indexdaten werden mit einem Verbund alle möglichen Wertekombinationen ermittelt und für diese der globale Ähnlichkeitswert in Abhängigkeit der lokalen Ähnlichkeitswerte und Attributgewichte zum ursprünglichen Datensatz berechnet. Attribut *a* geht mit einem Gewicht von *0,4*, die Attribute *b* und *c* jeweils mit einem Gewicht von *0,3* in die Gesamtähnlichkeit ein (diese Werte wurden für die Messung beliebig gewählt).

Ist die berechnete Gesamtähnlichkeit größer gleich *0,8*, so wird für diese Wertekombination von Attribut *a*, *b* und *c* ein Eintrag in der globalen Indextabelle *abcidx* zusammen mit der errechneten Ähnlichkeit und einer Referenz zum Datensatz erzeugt.

B.6.2 Suchanfragen

Zur Messung vergleichen wir die Ausführungszeiten drei verschiedener Suchanfragen. Zum einen stellen wir eine Anfrage nach einem Datensatz über die globale Indextabelle *abcidx*. Hier sind schon alle relevanten Informationen vorhanden, die gewünschte Mindestähnlichkeit von *0,8* wurde bereits beim Erzeugen der Indexdaten berücksichtigt.

Syntax 32 Suchanfrage über den globalen Index

```
SELECT tupleid, sim
FROM   abcidx
WHERE  avalue = 1625
  AND  bvalue = 3694
  AND  cvalue = 4188
ORDER BY 2 DESC
```

Die zweite Anfrage erfolgt über die lokalen Indexdaten der Tabellen *aidx*, *bidx* und *cidx*. Dazu wird ein Verbund über die Referenzen der Datensätze benötigt, die Gesamtähnlichkeit muß gemäß der Attributgewichte berechnet werden.

Syntax 33 Suchanfrage über die lokalen Indizes

```
SELECT a.tupleid, 0.4*a.sim + 0.3*b.sim + 0.3*c.sim AS sim
FROM aidx a, bidx b, cidx c
WHERE a.tupleid = b.tupleid
  AND b.tupleid = c.tupleid
  AND 0.4*a.sim + 0.3*b.sim + 0.3*c.sim >= 0.8
  AND a.value = 1625
  AND b.value = 3694
  AND c.value = 4188
ORDER BY 2 DESC
```

Die letzte Anfrage wird direkt auf der Datentabelle *abc* ohne Verwendung eines Index durchgeführt. Die Gesamtähnlichkeit wird für jeden Datensatz mit einer zuvor für jedes Attribut definierten Ähnlichkeitsfunktion berechnet.

Syntax 34 Suchanfrage mit Ähnlichkeitsfunktionen

```
SELECT id, 0.4*simInta(a,1625) + 0.3*simIntb(b,3694) + 0.3*simIntc(c,4188) AS sim
FROM   abc
WHERE  0.4*simInta(a,1625) + 0.3*simIntb(b,3694) + 0.3*simIntc(c,4188) >= 0.8
ORDER BY 2 DESC
```

B.6.3 Meßergebnisse

Die Meßergebnisse werden durch mehrere Messungen mit unterschiedlicher Anzahl von Datensätzen ermittelt. Bei jeder Messung werden jeweils fünf Anfragen nach unterschiedlichen Suchwerten durchgeführt, die Ausführungszeiten werden notiert.

10.000 Tupel. Die erste Messung erfolgt mit 10.000 Tupeln in der Tabelle *abc*. Dazu wurden in den lokalen Indextabellen *aidx*, *bidx* und *cidx* 130.000 Tupel, 110.000 Tupel und 90.000 Tupel erzeugt. Die globale Indextabelle *abcidx* enthält 630.000 Datensätze.

	Anfrage über globalen Index	**Anfrage über lokale Indizes**	**direkte Anfrage mit Ähnlichkeitsfunktionen**
Anfrage 1	0,41 sec	0,43 sec	6,30 sec
Anfrage 2	0,44 sec	0,44 sec	6,05 sec
Anfrage 3	0,43 sec	0,47 sec	6,18 sec
Anfrage 4	0,49 sec	0,43 sec	6,14 sec
Anfrage 5	0,43 sec	0,44 sec	6,20 sec
Durchschnitt	**0,44 sec**	**0,44 sec**	**6,18 sec**

15.000 Tupel. Die zweite Messung erfolgt mit 15.000 Tupeln in der Tabelle *abc*. Dazu wurden in den lokalen Indextabellen *aidx*, *bidx* und *cidx* 195.000 Tupel, 165.000 Tupel und 135.000 Tupel erzeugt. Die globale Indextabelle *abcidx* enthält 945.000 Datensätze.

	Anfrage über globalen Index	**Anfrage über lokale Indizes**	**direkte Anfrage mit Ähnlichkeitsfunktionen**
Anfrage 1	0,48 sec	0,49 sec	8,62 sec
Anfrage 2	0,46 sec	0,49 sec	9,19 sec
Anfrage 3	0,42 sec	0,42 sec	9,35 sec
Anfrage 4	0,44 sec	0,46 sec	9,16 sec
Anfrage 5	0,45 sec	0,43 sec	8,95 sec
Durchschnitt	**0,45 sec**	**0,46 sec**	**8,95 sec**

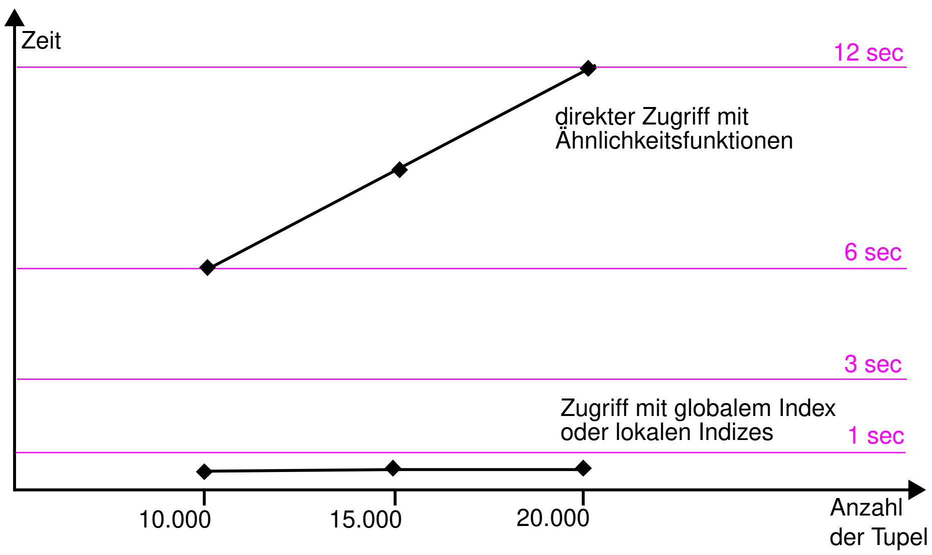
20.000 Tupel. Die letzte Messung erfolgt mit 20.000 Tupeln in der Tabelle *abc*. Dazu wurden in den lokalen Indextabellen *aidx*, *bidx* und *cidx* 260.000 Tupel, 220.000 Tupel und 180.000 Tupel erzeugt. Die globale Indextabelle *abcidx* enthält 1.260.000 Datensätze.

	Anfrage über globalen Index	**Anfrage über lokale Indizes**	**direkte Anfrage mit Ähnlichkeitsfunktionen**
Anfrage 1	0,43 sec	0,45 sec	12,13 sec
Anfrage 2	0,41 sec	0,43 sec	11,97 sec
Anfrage 3	0,41 sec	0,44 sec	12,02 sec
Durchschnitt	**0,41 sec**	**0,44 sec**	**12,02 sec**

	Anfrage über globalen Index	Anfrage über lokale Indizes	direkte Anfrage mit Ähnlichkeitsfunktionen
Anfrage 4	0,40 sec	0,42 sec	11,88 sec
Anfrage 5	0,41 sec	0,44 sec	12,11 sec
Durchschnitt	**0,41 sec**	**0,44 sec**	**12,02 sec**

Die folgende Abbildung verdeutlicht noch einmal die Ausführungszeiten der Suchanfragen in Abhängigkeit der Tupelanzahl in der Datentabelle.

Abbildung 7 Ausführungszeiten für globalen Index



Bei einer Datenmenge von 20.000 Tupeln benötigt eine Suchanfrage sowohl über die globale Indexstruktur als auch über die lokalen Indexdaten nur noch etwa 3,4 Prozent der Zeit, die zur direkten Suche auf der Datentabelle mit Aufrufen der Ähnlichkeitsfunktionen nötig ist.

Kapitel 7 Literaturverzeichnis

[BERG01] Dr. Ralph Bergmann, Universität Kaiserslautern
Experience Management: Foundations, Development Methodology and Internet-Based Applications
2001

[BRAD97] John Bradshaw, GlaxoWellcome Inc.
Introduction to Tversky similarity measure
Februar 1997

[DB2FTSP01] Albert Maier, Hans-Joachim Novak
DB2's Full-Text Search Products, White Paper
Juni 2001

[DB2TEAP00] IBM Corporation
DB2 Text Extender, Administration and Programming, Version 7
Juni 2000

[DELTA95] Computer Science Library 2000 TU Berlin
Arbeitsgruppe Delta: Suche mit Ähnlichkeitsmaßen in Datenbanken
November 1995

[GROHE99] Martin Grohe, Universität Freiburg
Isomorphism testing for embeddable graphs through definability
1999

[IFMXEX01] IBM Corporation
Informix Excalibur Text Search DataBlade Module, Version 1.3, User's Guide
Oktober 2001

[IFMXSQLa01] IBM Corporation
IBM Informix Guide to SQL - Syntax
August 2001

[IFMXSQLb01] IBM Corporation
IBM Informix Guide to SQL - Reference
August 2001

[KART02] M. N. Karthik, Moshe Davis
Search Using N-gram Technique Based Statistical Analysis for Knowledge Extraction in Case Based Reasoning Systems
März 2002

[MERK01] Dr. Rainer Merkl, Universität Göttingen
Heuristische Verfahren des Sequenzvergleichs
Mai 2001

[NARA00] National Archives and Records Administration NARA
The Soundex Indexing System
Februar 2000

[OTAD01] Oracle Corporation
Oracle Text, Application Developer's Guide, Release 9.0.1
Juni 2001

[OTFO01] Oracle Corporation
Oracle Text Features Overview
März 2001

[OTR01] Oracle Corporation
Oracle Text Reference, Release 9.0.1
Juni 2001

[OTWP01] Oracle Corporation
Oracle Text, An Oracle Technical White Paper
Mai 2001

[RR00] Raimund L. Feldmann, Birgit Geppert, Wolfgang Mahnke, Norbert Ritter, Frank Rößler
An ORDBMS-based Reuse Repository Supporting the Quality Improvement Paradigm
August 2000

[RR02] Wolfgang Mahnke, Norbert Ritter
The ORDB-based SFB-501-Reuse-Repository
März 2002

[SCHEI01] Hans-Jürgen Scheibl, FH für Technik und Wirtschaft Berlin
Grundlagen des phonetischen Datenbankabgleichs
Juni 2001

[SQL99a] American National Standard ANSI/ISO/IEC 9075-1:1999
Information Systems, Database Language - SQL
Part 1: Framework
Dezember 1999

[SQL99b] American National Standard ANSI/ISO/IEC 9075-2:1999
Information Systems, Database Language - SQL
Part 2: Foundation
Dezember 1999

[STONE99] M. Stonebraker, M. Brown
Object-Relational DBMSs - Tracking the Next Great Wave
1999