Humboldt Universität zu Berlin
Institut für Informatik

Diplomarbeit

Leistungssteigerung in Datenbanken durch maschinelles Lernen

FRANK HUBER

9. März 2004

Humboldt Universität zu Berlin

Institut für Informatik

Diplomarbeit

Leistungssteigerung in Datenbanken durch maschinelles Lernen

Frank Huber

Aufgabensteller und Betreuer: PROF. J.C.FREYTAG PH.D.

Abgabedatum: 25. Februar 2004

Erklärung:

Ich versichere, dass ich diese Diplomarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Berlin, den 25. Februar 2004

Ich erkläre, dass diese Diplomarbeit in der Zweigbibliothek für Informatik ausgelegt werden darf.

Berlin, den 25. Februar 2004

Abstrakt

Mit dieser Arbeit sollen die Möglichkeiten des maschinellen Lernens zur Leistungssteigerung in Datenbanksystemen untersucht werden. Dazu wird ein neues Konzept zur Selektivitätsabschätzung durch mehrdimensionale Statistiken vorgestellt. Dabei steht das Lernen der Statistiken durch Rückkopplung im Vordergrund. Im Konzept wird der Werteraum mit Hilfe einer Z-Ordnung diskretisiert und aufgezählt. Die einzelnen Klassen des so diskretisierten Werteraumes werden dann unter Benutzung von balancierten Bäumen abgespeichert und verarbeitet.

Inhaltsverzeichnis

Kapitel 1

Einführung

Im Jahr 1970 verfasste E.F. CODD in einem der IBM Research Laboratories ein Bericht über das Relationen-Modell und leitete damit den Start in eine neue Ära von Datenbankmanagementsystemen[1] ein[KS91]. Seitdem wurde auf dem Gebiet der Relationalen Datenbanken sehr viel geforscht. Die heutigen Datenbanksysteme[2] bestehen aus hoch komplexen Softwarekomponenten. Für die Ausschöpfung der maximalen Leistung des DBSs ist es notwendig, hochqualifizierte Datenbankadministratoren zu beschäftigen. Diese Administratoren sind entsprechend teuer und erhöhen somit die Kosten des Betreibens dieser DBSe. Daher versuchen die Hersteller von DBMSen die Kosten der Benutzung ihres Systems zu verringern. Dabei wird zum einen auf Werkzeuge zurückgegriffen, die die Administratoren bei diversen Entscheidungen unterstützen, zum anderen werden Aufgaben durch Automatisierung vereinfacht. Mit Hilfe dieser Neuerungen kann fast jeder sein System an ein gewisses Optimum von Leistung heran bringen. Dabei stellt sich die Frage, ob dieses Optimum auch schon das Maximum darstellt und, falls dies nicht der Fall ist, wie man sich diesem Maximum nähern kann. Ein Ansatz ist, internes Wissen des DBMSs zu nutzen. Dabei werden die Abläufe innerhalb des DBMSs überwacht. Nicht optimale Situationen werden gesucht und optimiert. Somit wird aus Fehlern der Vergangenheit gelernt. Mit dieser Arbeit soll untersucht werden,

[1] Datenbankmanagementsystem - DBMS
[2] Datenbanksystem - DBS

inwieweit sich die Möglichkeiten des maschinellen Lernens bezüglich der Leistungssteigerung in DBSen einsetzen lassen. Dabei sollen im 2. Kapitel die Grundlagen gelegt werden. In Kapitel 3 wird dann das Lernen in DBMSen vorgestellt. Anschließend soll in den Kapiteln 4, 5, 6 und 7 Lernen mit Hilfe von Statistiken vorgestellt und mein eigener Ansatz dargestellt werden. Im Kapitel 8 wird eine Zusammenfassung und ein Ausblick auf künftige Entwicklungen gegeben.

Kapitel 2

Grundlagen

Dieses Kapitel soll eine Einführung in das Gebiet des maschinellen Lernens, die Anfragebearbeitung sowie die Anfrageoptimierung geben. Im ersten Abschnitt werden dazu Definitionen aus verschiedenen Bereichen zum Begriff des Lernens beziehungsweise des maschinellen Lernens gegeben und miteinander verglichen. Desweiteren werden zwei Verfahren zum maschinellen Lernen aus dem Gebiet der Künstlichen Intelligenz vorgestellt. Der zweite Abschnitt wird sich dann mit der Anfragebearbeitung und Anfrageoptimierung in Relationalen Datenbanksystemen beschäftigen.

2.1 Maschinelles Lernen

2.1.1 Lernen im Allgemeinen

„Was bedeutet Lernen?" - Nun, ein Blick in die Literatur verschiedener Bereiche lässt den Schluss zu, dass Lernen ein schwer zu definierender Begriff ist. Der Begriff des Lernens wird in den verschiedenen Disziplinen der Wissenschaften unterschiedlich benutzt.

Das deutsche Wort "lernen", geht auf die gotische Bezeichnung für "ich weiß" (lais) und das indogermanische Wort für "gehen" (lis) zurück. Damit könnte Lernen als ein Prozess, bei dem ein Weg zurückgelegt und dabei Wissen erlangt wird, gedeutet werden. [Mie01]

Definitionen aus der Literatur

Brockhaus [Bro90]: Lernen, im vorwissenschaftlichen wie auch im wissenschaftlichen Sprachgebrauch, ist der relativ dauerhafte Erwerb, die Aneignung von Kenntnissen, Fertigkeiten, Fähigkeiten, Einstellungen und Verhaltensweisen oder ihre Änderung aufgrund von Erfahrung.

Pädagogik [Pae90]: Nach E.R. HILGARD und G.H. BOWER ist Lernen eine Veränderung im Erleben und Verhalten eines Individuums, die durch wiederholte Erfahrung in der Interaktion mit der Umwelt zustande kommt.

Philosophie [Phi90]: Learning is the acquisition of some true belief or skill through experience. - Lernen ist die Aneignung von Kenntnissen oder Fähigkeiten durch Erfahrungen.

Biologie [Bio90]: Learning and memory describe the abitity of organisms to benefit from prior experience. - Lernen und Gedächtnis beschreibt die Fähigkeit des Organismuses von vorherigen Erfahrungen profitieren zu können.

Medizin [Med90]: Aufnahme von Informationen zum Zweck der Reproduzierbarkeit, Stiften bedingter Reflexe, mit dem Ziel einer besseren Einpassung in die materielle und soziale Welt.

Eine Reihe von weiteren Definitionen werden in [Sta02] gegeben.

Vergleich der Definitionen

Der Vergleich der Definitionen führt zu einer gewissen Menge an Gemeinsamkeiten. Diese wären:

1. lernen ist ein Prozess, also eine Aktion über einen Zeitraum.
2. beim Lernen werden bereits gemachte Erfahrungen nutzbringend verwendet.

3. durch Lernen sollen Änderungen am Individuum, Objekt auftreten.

Eine Definition, die den Begriff des Lernens gut zu fassen und die Gemeinsamkeiten aus den verschiedenen Bereichen auf einen Punkt zu bringen scheint, wird in [TO02] gegeben. Dort heißt es:

> „Lernen ist jeder Prozess, durch den es zu einer dauerhaften Verhaltens- oder Wissensänderung eines Individuums aufgrund von Reizen, Signalen oder Situationen kommt."

2.1.2 Lernen auf dem Gebiet der Künstlichen Intelligenz

Im Buch „KI: Die Grundlagen" [Tan87] Kapitel 8, wird maschinelles Lernen als das Gebiet der KI[1] vorgestellt, das sich auf Verfahren zur Selbstoptimierung konzentriert. Demnach „lernen" Informationsverarbeitende Systeme, wenn sie ihre Arbeitsweise verbessern oder ihre Wissensbasen vergrößern.

> „Während des Lernens vertieft ein System sein Wissen und seine Fähigkeit, eine oder mehrere Aufgaben auszuführen. Die Vertiefung resultiert aus der Tätigkeit, Informationen zu verarbeiten. Sie kann auf die verschiedenste Art und Weise geschehen ... . Die Verbesserung besteht jedoch meist in irgendeiner Form der Leistungssteigerung."

MECHLER gibt in seinem Buch [Mec95] „Intelligente Informationssysteme" eine etwas andere Definition.

> „Maschinelles Lernen bezeichnet Veränderungen in Systemen, die anpassungsfähig sind, in dem Sinne, dass diese Systeme in der Lage sind, die gleichen oder ähnliche Probleme in einer wiederkehrenden Situation besser zu lösen."

Die folgenden Punkte lassen sich als Ziele des maschinellen Lernens ansehen:

[1] KI - Künstlichen Intelligenz

- die Verbesserung der Leistungsfähigkeit eines Systems
- die Realität adäquat abzubilden
- Wissenserwerb und Erweiterung des bereits vorhandenen Wissens

Formale Definition von Lernen

Sei P ein informationsverarbeitendes System und M eine Funktion, die den "Wert" von P's Berechnungen oder Zuständen misst. Wir sagen dann, dass P hinsichtlich M über einen Zeitraum $[t_1, t_2]$ lernt, wenn der Wert $M(P)$ zum Zeitpunkt t_2 größer ist als der Wert $M(P)$ zum Zeitpunkt t_1 und sich die Veränderung aus der Informationsverarbeitung von P ergibt. [Tan87, Kap 8, S.333]

Lernen kann somit als monotone Bewegung von einer Stufe des Wissens oder der Erkenntnis auf höhere Ebenen begriffen werden. Im Gegensatz dazu spricht man gewöhnlich von Vergessen und Verlernen.

2.1.3 Verfahren des maschinellen Lernens

Verfahren zum automatischen oder maschinellen Lernens in der KI, lassen sich nach verschiedenen Kriterien klassifizieren. Eine Art der Klassifizierung ist die Einteilung in die zwei Gruppen der numerischen und strukturellen Lernverfahren. Andererseits ist die Gruppierung nach analytischen und synthetischen Lernverfahren möglich. Dabei umfassen numerische Verfahren die Algorithmen, die mit Schwellwerten oder Koeffizienten arbeiten. Strukturelle Verfahren dagegen arbeiten mit Algorithmen, die sich mit Bäumen, Graphen, Mengen und Relationen beschäftigen. Analytische und synthetische Verfahren werden durch die Fragestellung, ob neues Wissen erzeugt oder bereits vorhandenes Wissen in eine andere Form gebracht werden soll, unterschieden. [Mec95, Tan87]

MICHALSKI gibt in [KM90] ein weiteres und allgemein akzeptiertes Klassifikationsschema an, das MECHLER in [Mec95, Seite 50/51] präsentiert. Dabei werden Verfahren wie folgt unterteilt:

Auswendiglernen ist die niedrigste Stufe des Lernens. Hierbei ist die bloße Erinnerung an Fakten gefordert, aber keinerlei Schlussfolgerungen aus ihnen. Anwendungen für maschinelles Lernen dieser Stufe sind einfache Datenbanksysteme. Die Fakten oder Daten werden in geeigneter Form an das DBS übergeben, welches dann die Daten aufnimmt und abspeichert. Anfragen können dann mit den bereits vorhandenen Daten beantwortet werden.

Lernen mit Hilfe von Anweisungen ist die nächste Stufe. Hierbei müssen die übergebenen Fakten noch in eine interne, *nutzbringende* Darstellung transformiert werden. Ein Beispiel ist ein Mensch, der ein Buch liest. Im Buch werden eine Vielzahl von Fakten übergeben. Diese müssen aber erst in eine entsprechende Form gebracht werden, um sie nutzbringend abzuspeichern.

Deduktives Lernen Bei der Deduktion wird aus dem Allgemeinen das Besondere und Spezielle abgeleitet. Die Eingabe wird so transformiert, dass der Wahrheitsgehalt der Aussage erhalten bleibt. Die Ausgabe soll dann eine Verbesserung und Verfeinerung des ursprünglichen Wissens darstellen. Somit wird auf dieser Stufe des Lernens ein erhöhter Grad an Verantwortung für die Transformation des Wissens gefordert. Die Schlussfolgerungen aus dem vorhandenem Wissen werden in Form von Regeln formuliert.

Lernen aus Analogie ist die Form des Lernens, bei der eine Kombination von deduktiven und induktiven Ansätzen benutzt wird. Hierbei wird im ersten Schritt mit Hilfe der Induktion, ein, der Problemstellung ähnlicher Problembereich in der Wissensbasis gesucht. Der zweite Schritt ist die Lösung des aktuellen Problems durch Deduktion. Also die Spezialisierung des gefundenen Problembereichs bezogen auf die aktuelle Aufgabe.

Induktives Lernen ist die höchste Stufe des Lernens und fordert den größten Grad an eigenen Schlussfolgerungen. Induktion ist der Gegensatz zur Deduktion. Bei der Induktion wird aus dem Speziellen

auf das Allgemeine geschlossen. Induktives Lernen wird noch in folgende Unterkategorien unterteilt:

- Lernen aus Beispielen
- Lernen durch Experimentieren
- Lernen durch Entdecken

Als nächstes sollen zwei konkrete Verfahren vorgestellt werden, die zur Implementation von maschinellem Lernen benutzt werden können. Diese sind neuronale Netze und Entscheidungsbäume. Darüber hinaus gibt es noch viele andere Verfahren. Diese sollen aber in dieser Arbeit nicht weiter betrachtet werden.

Neuronale Netze

Neuronale Netze spielen eine wichtige Rolle im Bereich des maschinellen Lernens [Mec95]. Die Vorlage für neuronale Netze sind biologische Neuronen, wie sie im menschlichen Gehirn vorkommen. Das menschliche Gehirn besteht etwa aus 10^{11}Neuronen, die als Funktionseinheiten parallel arbeiten und hochgradig miteinander vernetzt sind.

In Neuronalen Netzen der Informatik werden die Neuronen durch nichtlineare Elemente dargestellt, die stark miteinander verbunden sind. Die Verbindungen werden durch veränderbare Gewichtungen beeinflusst. Das Wissen eines Neuronalen Netzes ist somit in seinen Vernetzungen und Gewichten der Verbindungen gespeichert. Durch das maschinellen Lernen werden Verbindungen und ihre Gewichte verändert. Es soll nun ein Beispiel eines neuronalen Netzes gegeben werden, Bild 2.1. Dabei soll eine Bool'sche Funktion mit Hilfe des Netzes darstellt werden. Bool'sche Funktionen sind Funktionen, bei denen der Werte- und Definitionsbereich nur zwei Elemente enthält, meist repräsentiert durch $\{0, 1\}$. Eine Bool'sche Funktion $f(x_1, x_2, ..., x_n)$ bildet somit ein n-Tupel von $(0, 1)$ auf $\{0, 1\}$ ab. In unserem Beispiel soll die Funktion $f(x_1, x_2, x_3, x_4) = x_2\neg x_3 + \neg x_1 x_3 x_4$ dargestellt werden. Die Kreise entsprechen den Funktionseinheiten. Die

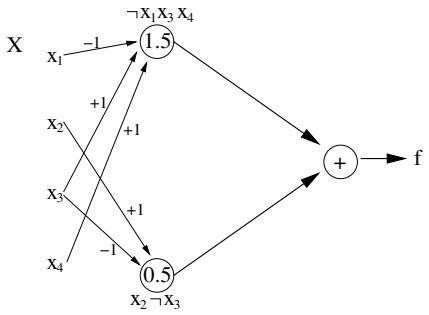

Abbildung 2.1: Beispiel Neuronales Netzwerk

beiden Funktionseinheiten der ersten Stufe bilden die jeweiligen Teilfunktionen, indem sie ihre Eingänge mit den Gewichten$(-1, +1)$ multiplizieren und dann zusammen addieren. Die Zahl innerhalb des Kreises gibt den Grenzwert an, bei dem die Funktionseinheit mit einem Ausgang von 0 oder 1 reagiert. Das Symbol$+$ steht für die Disjunktion der beiden Eingänge.

Entscheidungsbäume

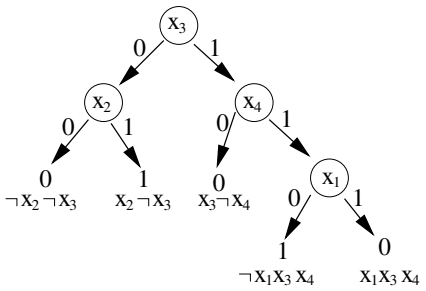

Abbildung 2.2: Beispiel Entscheidungsbaum

Entscheidungsbäume sind eine andere Art der Wissensrepräsentation. Wie das Wort schon sagt, sind es Bäume, bei denen die internen Knoten gegen Eingabemuster getestet werden und die Blätter Kategorien darstellen. Somit weist ein Entscheidungsbaum einem Eingangsmuster eine Kategorie zu, indem das Muster durch die einzelnen Tests gefiltert wird. Die bekanntesten Systeme, um Entscheidungsbäume zu lernen, sind ID3 und C4.5 von Quinlan [Qui93, Nil96]. Nun wollen wir die Bool'sche Funktion aus dem Beispiel der Neuronalen Netze nochmals mit Hilfe von einem Entscheidungsbaum darstellen, Bild 2.2. Dabei werden alle Muster, die nicht im Baum vorhandenen sind, in die Kategorie 0 ein-

geordnet.

2.2 Anfragebearbeitung in DBMSen

Die Anfragebearbeitung und Anfrageoptimierung in einem DBMS ist der Bereich, in dem aus einer deklarativen Anfrage ein ausführbares Programm erzeugt wird, welches dann die Daten aus der Datenbank extrahiert. Dieses Programm, Anfrageausführungsplan (engl. Query Execution Plan), meist QEP genannt, muss semantisch äquivalent zur Anfrage sein und sollte möglichst optimal sein. Im ersten Teil dieses Abschnitts soll ein Einblick in diesen Prozess, der Erzeugung des QEP's, gegeben werden. Anschließend sollen Leistung und Leistungsverhalten im Kontext von DBMSen erläutert werden.

2.2.1 Von der Anfrage zum Ausführungsplan

Das Gebiet der Anfragebearbeitung beinhaltet eine Menge von Aktivitäten, um Daten aus der Datenbank zu extrahieren. Auf Bild 2.3 ist die grobe Abfolge der einzelnen Aktionen dargestellt [KS91, CBS95, CG93, Fre02, Fre00, GLSW93].

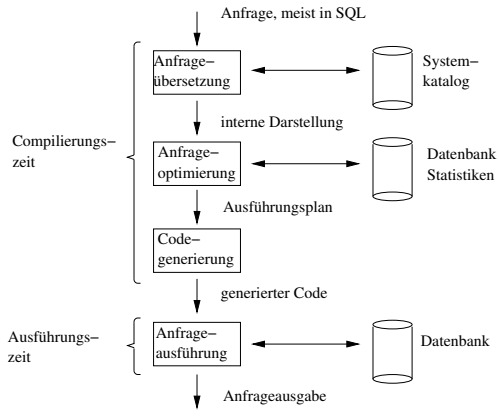

Abbildung 2.3: Anfragebearbeitung

Als erstes wird die Anfrageübersetzung durchgeführt. Der Zweck der Übersetzung ist als erstes die syntaktische und semantische Korrektheit der Anfrage sicher zu stellen. Danach wird die Anfrage in eine interne Repräsentation, zum Beispiel eine erweiterte Relationale Algebra oder QGM[2][Fre02], überführt. Im zweiten Schritt, der Optimierung, wird die interne Darstellung optimiert und in einen Ausführungsplan übersetzt. Bei der Optimierung werden zwei unter-

[2]Query Graph Modell

schiedliche Techniken benutzt. Die erste Technik benutzt Heuristiken, um die Operationen der Anfrage zu ordnen. Die andere Technik verwendet ein Kostenmodell, um die unterschiedlichen Ausführungspläne vergleichen zu können. Anhand dieser berechneten Kosten wird dann die günstigste (optimale) Variante ausgewählt. Zur Kostenberechnung im DBMS von IBM, DB2, wird z.B. eine Funktion genutzt, die linear von den Kosten für Ein- und Ausgabeoperationen, den CPU-Kosten und Kommunikationskosten des Plans abhängt [RZML02]. Zur Abschätzung der einzelnen Kosten werden dabei physische Eigenschaften der Relationen und andere Informationen, wie z.B. über vorhandene Indexe, genutzt. Die Kostenabschätzungen für einen Anfrageplan hängen sehr stark von den Abschätzungen der Selektivitäten, der einzelnen Operationen, ab. Die Selektivität eines Operator wird mit dem Verhältnis der Anzahl von ausgehenden Tupeln zur Anzahl der eingegangenen Tupeln bestimmt. Die entsprechenden Mengen müssen bei der Anfrageoptimierung geschätzt werden. Dabei wird meist mit Datenbankstatistiken gearbeitet, wie z.B. die Anzahl der Tupel einer Relation. Andere Abschätzungen, die nicht mit Hilfe von Statistiken berechnet werden können, unterliegen Modellannahmen, z.B. Gleichverteilung innerhalb einer Relation oder die Unabhängigkeit von Attributen einer Relation. Statistiken über Verteilungen werden meist mit Hilfe von Histogrammen dargestellt. Die Erstellung und Wartung der Histogramme sind zu meist kostenintensive Prozesse und sollten immer minimiert werden.

Als nächstes soll ein Beispiel zur Berechnung der Kosten für eine Anfrage geben werden. Die SQL-Anfrage soll folgendermaßen lauten:

```
Select * from X,Y,Z
Where X.Preis >= 100 and Y.Monat = 'Dez'
and Z.Stadt = 'Berlin'
and X.ID = Y.ID and Y.Nr = Z.Nr
Group By A;
```

Auf Bild 2.4 ist ein Ausführungsplan mit den einzelnen Operatoren für diese Anfrage abgebildet. Dabei bedeuten:

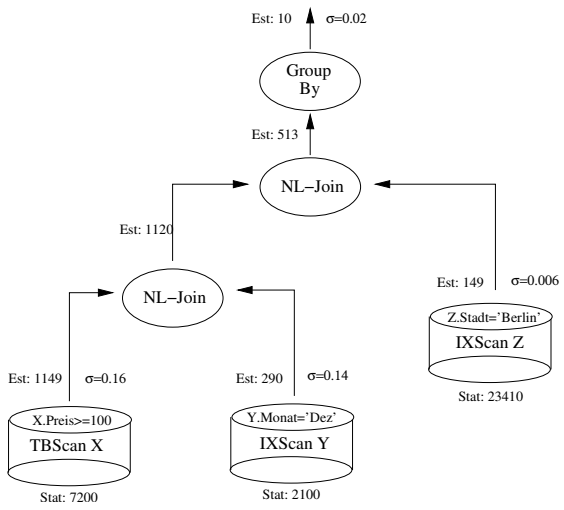

Abbildung 2.4: Anfrageplan und Kostenberechnung, Quelle [SLMK01]

Est ist die geschätzte Anzahl von Tupeln am Ausgang des Operators

Stat ist die Anzahl aller Tupel der jeweiligen Relation

σ ist die Selektivität des Selektionsoperators

TBScan steht für einen TableScan, der die gesamte Tabelle einliest und mit dem Prädikat *X.Preis>=100* filtert

IXScan steht für einen IndexScan, bei dem ein Index genutzt wird, um das jeweilige Prädikat zu erfüllen

NL-Join steht für einen Nested-Loop-Join

Group By stellt den Operator für die Funktion von *Group By A* in der Anfrage dar

Die Gesamtkosten des Plans werden dann mit Hilfe der Selektivitäten oder Kardinalitäten aus den Operatoren bestimmt.

2.2.2 Leistung und Leistungsverhalten eines DBMSs

Was ist Leistungsverhalten in einem Datenbanksystem? Das Leistungsverhalten einer Anfrage an das DBMS kann als die Antwortzeit ($t_{Anfrage}$) im Verhältnis zu Betriebsmittelverbrauch ($R_{Anfrage}$), also $\frac{1}{t_{Anfrage} * R_{Anfrage}}$ mit $0 < t_{Anfrage}$und $0 < R_{Anfrage} \leq 1$ definiert werden. Bei einer optimalen Systemkonfiguration würde also die Antwortzeit und der Betriebsmittelverbrauch jeweils ein Minimum annehmen. Es ist klar, dass die Antwortzeit von dem Betriebsmittelverbrauch abhängt. Daher wird der Nutzer zwischen Zeit und Verbrauch wählen müssen, es ergibt sich also ein Optimierungsproblem. In den meisten Systemen werden gleichzeitig mehrere Anfragen vom DBMS bearbeitet. Somit sollte das Leistungsverhalten des DBSs als eine Kombination aus der Anzahl der Anfragen, dem Betriebsmittelverbrauch und der Antwortzeit definiert werden. Es muss eine maximale Anzahl von Anfragen bei möglichst geringer Antwortzeit bearbeitet werden. Dazu muss versucht werden, die Betriebsmittel des Systems zu einem Maximum zu nutzen, ohne eine Ressource zu überlasten. [SCF+86, KS91, CBS95, CG93]

Ursachen für schlechtes Leistungsverhalten Es existieren eine Vielzahl von mögliche Ursachen. In [CG93] werden vier Hauptgruppen von Ursachen angegeben. Diese sind:

1. DATENBANK-ENTWURF: Entwurfsprobleme entstehen im Allgemeinen durch Analysten und Designer, die nicht:

   - das Leistungsverhalten bei dem Entwurf des Datenmodells beachtet haben,
   - Programme entworfen haben, wie es für eine relationale Datenbank nötig gewesen wäre,
   - Programme entworfen haben, wie es für die gegebene Hardwarekonfiguration nötig gewesen wäre.

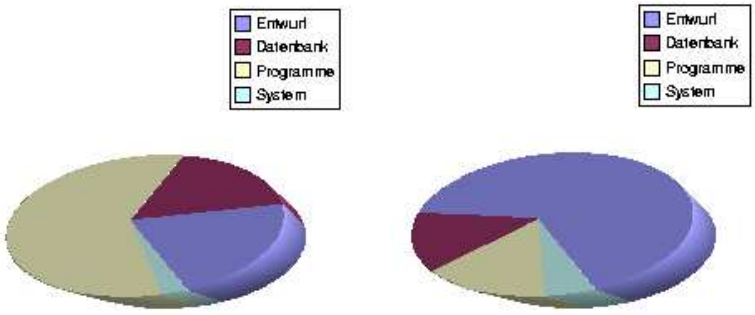

Abbildung 2.5: (a) Ursachen für schlechte Antwortzeiten, (b) Aufwand zur Lösung, Quelle [CG93]

2. PROGRAMME: Programmprobleme entstehen meist durch Programmierer, die SQL-Statements genutzt haben, die der Optimierer nicht effizient nutzen kann

3. DATENBANK: Datenbankprobleme werden häufig durch Datenbankadministratoren verursacht, die :

   - die Betriebsmittel der jeweiligen Maschine nicht effizient nutzen ,
   - die Datenbank nicht für eine optimale Sekundärspeichereingabe und -ausgabe strukturiert haben.

4. SYSTEM: Systemprobleme entstehen durch:

   - ein nicht entsprechend optimiertes Betriebssystem,
   - Maschinen, die für das entsprechende DBMS nicht adäquat ausgerüstet sind.

Nach [CG93] sind die vier Gruppen, wie auf Bild 2.5(a) abgebildet, an den Problemen beteiligt und können mit dem entsprechenden Aufwand (Bild 2.5 (b)) gelöst werden.

Weitere Ursachen Andere Ursachen liegen im DBMS selbst. Es kann durchaus vorkommen, dass der erzeugte Anfrageausführungsplan nicht optimal ist oder einfach unter falschen Annahmen berechnet wurde. Wenn z.B. die Datenbankstatistiken, die bei der Kostenberechnung verwendet wurden, nicht mehr die Verteilungen der Daten in der Datenbank widerspiegeln, kann es zu schlechtem Leistungsverhalten kommen, obwohl vielleicht gar kein Fehler aus den oberen vier Kategorien vorliegt. Weiter können auch Annahmen, die in den meisten Optimierern als Grundlage dienen, für die gegebene Datenbank falsch sein. Es wird z.B. oft von einer Gleichverteilung ausgegangen, wenn keinerlei Information über die eigentliche Verteilung vorliegt. Andere Fehlerquellen sind Annahmen, wie die Unabhängigkeit der Attribute einer Relation. Ein konkretes Beispiel hierzu:
Es sei die Anfrage *select * from autoteile where hersteller='FOO' and vertrieb ='BAR'* gegeben. Die Relation *autoteile* besteht aus den Attributen *teilID, hersteller* und *vertrieb*, wobei die *teilID* ein Teil eindeutig identifiziert, *hersteller* den Hersteller beschreibt und *vertrieb* denjenigen angibt, der das Teil verkauft.

Es sollen folgende Annahmen gelten:

1. die Anzahl aller Tupel der Relation *autoteile* ist 1000.
2. die Anzahl der Tupel mit *hersteller='FOO'* ist 100.
3. die Anzahl der Tupel mit *vertrieb='BAR'* ist 500.
4. Der Hersteller *FOO* kauft seine Teile nur über den Vertrieb *BAR*.

Die Selektivität der Anfrage würde somit wie folgt berechnet werden:

Mit Annahme der Unabhängigkeit

$$\sigma_{hersteller='FOO'}(autoteile) * \sigma_{vertrieb='BAR'}(autoteile) = \frac{100 * 500}{1000000} = \frac{5}{100}$$

Mit Wissen über Annahme 4

$$\sigma_{hersteller='FOO'}(autoteile) = \frac{100}{1000} = \frac{1}{10}$$

Somit ist die Selektivität ohne das Wissen der Annahme 4, um etwa 10 mal höher als die Selektivität mit dieser Annahme. Solche Unterschiede führen dann zu unterschiedlichen Kosten und damit zu unterschiedlichen Plänen.

Kapitel 3

Lernen in Datenbankmanagementsystemen

In diesem Kapitel sollen Möglichkeiten des Lernens in DBMSen vorgestellt werden. Dazu soll im ersten Abschnitt eine Aufteilung in Teilbereiche des Lernens vorgenommen werden. Am Ende werden die einzelnen Teilbereiche auf mögliche Zusammenhänge geprüft. Im zweiten Abschnitt werden bereits veröffentlichte Projekte vorgestellt, die in das Gebiet des Lernens in DBSen fallen.

3.1 Kategorisierung des Lernens in DBMSen

In diesem Abschnitt soll das Lernen in DBMSen in vier verschiedene Kategorien einteilt werden. Die Kategorien wurden so gewählt, dass sie sich so wenig wie möglich überschneiden und natürlich das Ziel der Leistungssteigerung verfolgen.

Die im folgenden vorgestellten vier Kategorien sind:

- Lernen über Relationen und die physische Datenstruktur
- Lernen bezüglich der Anfragen, die an das DBMS gestellt werden
- Lernen über die zur Verfügung stehenden Betriebsmittel

- Lernen über die Topologie eines verteilten DBSs

3.1.1 Lernen über Relationen und die physische Datenstruktur der Datenbank

Im Bereich des Lernens über Relationen stehen sicherlich Statistiken im Vordergrund. Denn nur mit den passenden, korrekten und aussagekräftigen Statistiken lassen sich Anfrageausführungspläne erzeugen, deren Kostenberechnungen mit den Kosten der Ausführung äquivalent sind. Und diese Äquivalenz ist die Grundlage für die richtige Wahl des Ausführungplans. Im Bereich der physischen Datenstrukturen können z.B. bei der Anordnung der Daten auf dem Sekundärspeicher und bei der Verteilung der Daten über ein Netzwerk Leistungsgewinne erzielt werden. Da speziell die Ein- und Ausgabeoperationen des Sekundärspeichers und des Netzwerkes die Kosten für eine Anfrage dominieren. Somit führen Leistungsverbesserungen in diesen Bereichen zum Teil zu deutlichen Verbesserungen im Leistungsverhalten des DBSs. [AC99, CN98a, KW99, VZZ+00, RZML02, ML02, SLMK01]

3.1.2 Lernen bezüglich der Anfragen, die an das DBMS gestellt werden

In vielen kommerziellen Datenbanksystemen existieren Werkzeuge, um die verschiedensten Entscheidungen durch das Datenbanksystem treffen zu lassen. Beispiele sind der INDEX-ADVISOR, CONFIGURATION-ADVISOR von IBM und INDEX-WIZARD von MICROSOFT. All diese Werkzeuge haben gemeinsam, dass sie für ihre Analysen und Berechnungen eine Menge von Anfragen (engl. Workload) und deren Häufigkeit der Abarbeitung im System benötigen. Nun soll der Workload repräsentativ für eine Systemkonfiguration sein. Hier wäre es möglich, diesen Workload aus dem laufenden System zu ermitteln, zu lernen. Der Ansatz ist bereits in den kommerziellen Systemen insofern integriert, dass es möglich ist, die Anfragen über die Zeit zu überwachen, und dann die häufigsten und teuersten zu

ermitteln, und als Workload abzuspeichern. Auch lässt sich Wissen aus Anfragemustern, und natürlich den Anfragen selbst, erlernen. Vorstellbar ist z.B., aus den Anfragemustern dynamische Cachestrategien zu entwickeln oder abzuleiten. [DR99, CGN02, CN98b, VZZ+00, CN00, CN97]

3.1.3 Lernen über die zur Verfügung stehenden Betriebsmittel

Nach [CG93] wird das Leistungsverhalten eines Computers von der Leistung der folgenden vier Komponenten hauptsächlich beeinflusst:

- Speicher
- Festplatten-Ein-/Ausgabe
- Prozessor
- Netzwerk

Dabei sind die Leistungen der einzelnen Bereiche nicht unabhängig von einander. Ein Beispiel hierzu: Wenn in einem System nur wenig Speicher vorhanden ist, wird bei der Überlastung dieses Betriebsmittels, die Verwendung von Swapping oder Paging notwendig werden. Swapping und Paging sind Begriffe, um die Auslagerung von Daten aus dem Speicher auf die Festplatte zu beschreiben (siehe [Fri91], Kap.7, Speicherverwaltung). Durch diese Auslagerung werden zusätzliche Prozessorschritte und Festplatten-Ein-/Ausgabeoperationen notwendig. Und somit werden diese beiden Betriebsmittel stärker belastet, als wenn ein größerer Speicher vorhanden wäre. Daher ist der Versuch ein System an sein Maximum an Leistung zu bringen, immer der Versuch, aus jeder Komponente das Maximum herauszuholen, ohne eine der Komponenten dabei zu überlasten. In [CG93] wird das Einstellen von Systemen (engl. tuning) mit den folgenden Punkten beschrieben:

- Ausnutzung des Speichers bei nahezu 100%,

- die Menge von Festplattenzugriffen gleichmäßig über alle vorhandenen Platten zu verteilen und diese innerhalb der empfohlenen maximalen Ein-/Ausgabe-Raten laufen zu lassen,
- der Prozessor sollte während Spitzenzeiten beinahe zu 100% ausgelastet sein, ohne das Benutzerprogramme warten müssen,
- der Netzwerkverkehr sollte wenig unterhalb der angeratenen maximalen Belastungen sein und keine Kollisionen verursachen,
- eine minimale Benutzung von Paging und Swapping Mechanismen.

Die Aufgabe des Optimierens eines Systems bedarf somit einer ständigen Kontrolle, Überwachung und eventuellen Veränderungen. Somit kann auch hier die Möglichkeit des Lernens in DBMSen eine Leistungssteigerung bedeuten, wenn die aktuellen Ausführungen überwacht und nach entsprechenden Punkten immer weiter optimiert werden. Es ist zum Beispiel bei einer Sortieroperation denkbar, die Größe des Speicherplatzes für die Sortieroperation dynamisch zu bestimmen, um ein Optimum an Speicherauslastung zu erhalten. Ein anderer Anwendungsfall ist eine verteilte Umgebung. Dort ist es für den Optimierer notwendig, die aktuellen Betriebsmittel, der zur Verfügung stehenden Knoten zu kennen, um den beziehungsweise die richtigen Knoten zur Bearbeitung der Anfrage auswählen zu können. Dafür wäre es notwendig, die Betriebsmittel zu überwachen und die Informationen in entsprechender Form dem Anfragebearbeiter zur Verfügung zu stellen. Aus dieser Überwachung können dann im nachhinein Informationen bezüglich falscher Lastverteilung oder Systemkonfigurationen gewonnen werden. Somit sind bei der Systemüberwachung zwei verschiedene Methoden bzgl. der Zeit zu unterscheiden. Die eine ist die aktuelle Ermittlung der Betriebsmittel, um den Anfragebearbeiter in seiner Entscheidung für den richtigen Plan zu unterstützen. Dieses entspricht somit immer einer Momentaufnahme der aktuellen Betriebsmittel. Die andere Methode ist die, bei der eine Überwachung über einen Zeitraum geschieht. Die sich daraus ergebenden Verläu-

fe können dann auf Probleme, wie falsche Lastverteilung, geprüft werden. [Fri91, Lou90, Hun98]

3.1.4 Lernen über die Topologie eines verteilten DBSs

In verteilten DBSen können die verschiedenen Komponenten des DBMSs auf einem oder mehreren Rechnern laufen. Als die Topologie des DBSs sollen die Beziehungen der einzelnen Komponenten untereinander verstanden werden. Die Topologie kann mit Hilfe von beschrifteten, ungerichteten Graphen dargestellt werden, wobei die Knoten die einzelnen Komponenten darstellen, und die Kanten die Verbindungen zwischen den Komponenten bilden. Diese Verbindungen können sowohl lokale Verbindungen sowie Verbindungen über ein physisches Netzwerk sein. Die Kanten dieser Graphen können dann mit verbindungsspezifischen Werten beschriftet werden, z. B. mit der Art des Netzwerkes (lokal,Ethernet,...), der Geschwindigkeit oder durchschnittliche Kapazitäten. Die Knoten könnten mit Hilfe der einzelnen Komponenten im Zusammenhang mit dem entsprechenden Rechner beschriftet werden. Daraus würde sich dann folgende Frage ergeben: „Was kann aus solchen Graphen gelernt werden?"

Es kann zum Beispiel abgelesen werden, wieviele und welche Komponenten im System agieren und daraus schließen, ob es notwendig ist, neue Komponenten zu starten oder eventuell zu stoppen. Ebenfalls ist die Ablehnung von Anfragen vorstellbar, falls festgestellt wurde, dass das System voll ausgelastet ist. Somit könnte das DBMS erst einmal alle bereits laufenden Anfragen beantworten. Dieses würde somit zu einer Entlastung des Systems führen.

3.1.5 Zusammenhänge der einzelnen Kategorien

Die vier Gebiete sind bei vielen Situationen des Lernens in DBMSen nicht unabhängig voneinander. Sie verschmelzen vielmehr und gehen ineinander über. Dieses soll am Beispiel des Lernens in Bezug auf die optimale Menge von Indexen im System verdeutlicht werden. Dabei werden die

Kategorien „Lernen über Relationen und die physische Datenstruktur der Datenbank", „Lernen bezüglich der Anfragen, die an das DBMS gestellt werden" und „Lernen über die zur Verfügung stehenden Betriebsmittel" benötigt. Bei dieser Optimierung sollten die folgenden Punkte beachtet werden:

- die physische Datenorganisation,
- die Optimierung bzgl. einer repräsentativen Menge von Anfragen,
- die vorhandenen Betriebsmittel, da nur eine begrenzte Menge von Platz auf dem Datenträger zur Verfügung steht.

3.2 Projekte aus Forschung und Wirtschaft

In den letzten Jahren wurden mehrere Projekte durchgeführt, die zu mehr Autonomie in DBMSen führen sollen. Dabei sind die Projekte SMART von IBM und AUTOADMIN von MICROSOFT beispielgebend. Die Abkürzung SMART steht dabei für Self-Managing And Resource Tuning. Beide Projekte versuchen, die Aufgaben des Datenbankadministrators (DBA) zu vereinfachen oder zu automatisieren. Sie bestehen aus einer Vielzahl von Teilprojekten (siehe [ZLLL01, Mir03, Hub03]). Im folgenden sollen einzelne Arbeiten aus den Projekten vorgestellt werden, die bezüglich des Lernens in DBMSen wichtig sind. Dabei sollen diese Arbeiten mit Hilfe der Kategorisierung aus dem Abschnitt 3.1 eingeordnet und etwas abstrakter und allgemeiner dargestellt werden.

3.2.1 LEO - DB2's LEarning Optimizer

Das Projekt LEO von IBM, ist wie der Name schon sagt, ein lernender Optimierer. Wie im Abschnitt 2.2 vorgestellt, liegt vielen Anfrageoptimierern moderner DBMSe ein Kostenmodell zu Grunde. Mit Hilfe dieses Modells werden die Kosten einer Anfrage und damit seines Anfrageausführungsplans bewertet, um dann den kostengünstigsten Plan auszuwählen. Diese

Kosten sind aber auch immer an Modellannahmen geknüpft und können sich somit bei falschen Annahmen oder fehlerhaften Statistiken stark von den realen Kosten bei der Ausführung unterscheiden. Mit Hilfe von LEO sollen solche fehlerhaften Berechnungen erkannt und nach den Ursachen gesucht werden, sodass bei später folgenden ähnlichen Anfragen, die Fehler korrigiert werden können und ein besserer Anfrageausführungsplan ausgewählt wird.

Im Bild 3.1 ist der normale Ablauf der Anfragebearbeitung und Ausführung ohne Lernen abgebildet. Die Anfrage wird bearbeitet, ein Anfrageplan (QEP) erstellt und mit Hilfe der Anfrageausführungseinheit (QEE)[1] ausgeführt. Der QEP besteht dabei aus n Teilschritten. Zur Berechnung der Kosten des Plans wurde die Menge S von Statistiken genutzt. Die Kosten der Einzelschritte, ϕ_i, ist eine Funktion f mit dem Parameter S, also $\phi_i = f(S)$. Die Gesamtkosten, $Cost$, des QEPs ist eine Funktion g mit $\phi_i, 1 \leq i \leq n$, als Parameter. $Cost = g(\phi_1, ..., \phi_n)$

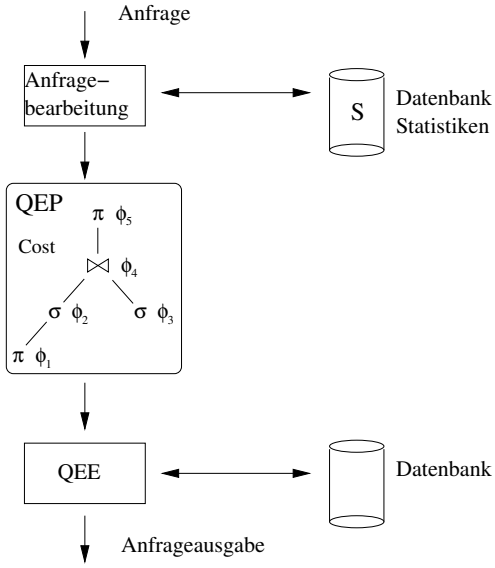

Abbildung 3.1: Anfragebearbeitung ohne LEO

Auf dem Bild 3.2 ist der Ablauf mit Lernen dargestellt. Auch hier wird die Anfrage bearbeitet, ein QEP erstellt und mit Hilfe der QEE ausgeführt. Zur Berechnung der Kosten des Plan werden hier die Menge S von Statistiken und eine Menge Δ von Anpassungen zu den Statistiken genutzt. Die Kosten der Einzelschritte ϕ_i ist dann eine Funktion f' mit dem Parametern S und Δ , also $\phi_i = f'(S, \Delta)$. Die Gesamtkostenfunktion, g, bleibt unverändert, $Cost = g(\phi_1, ..., \phi_n)$. Während der Ausführung werden dann die einzelnen Selektivitäten, ϕ_{Ri}, der Teilschritte gemessen und gespeichert. Mit Hilfe der gemessenen und der berechneten Selek-

[1] Query Execution Engine

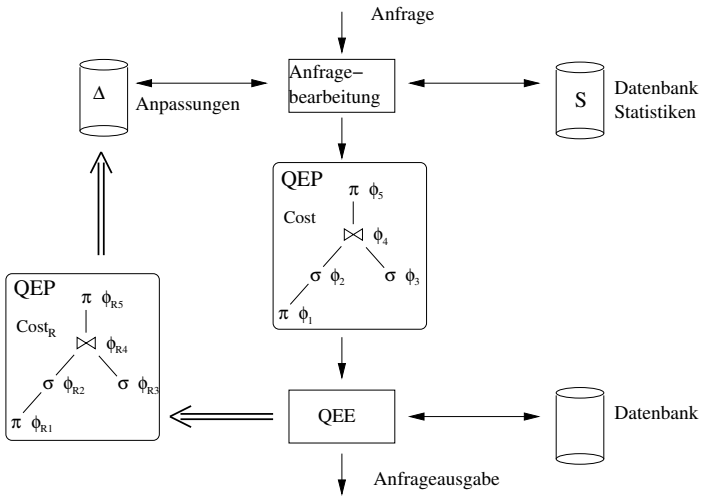

Abbildung 3.2: Anfragebearbeitung mit LEO

tivitäten, (ϕ_{Ri}, ϕ_i), können dann die Anpassungen, Δ, berechnet werden, $\Delta = \Delta \oplus h(\phi_1, ..., \phi_n, \phi_{R1}, ..., \phi_{Rn})$. Der $\oplus$ Operator wird benutzt, um die alten und die neu berechneten Anpassungen miteinander zu vereinen.

In [SLMK01, ML02] werden Anpassungen und Statistiken getrennt gespeichert. Dadurch müssen bei jeder Kostenberechnung immer die Anpassungen und die Statistiken zusammengerechnet werden. Der Hauptvorteil ist, dass mit dieser Methode das Lernen oder die LEO Komponente einfach auszuschalten ist.

Bei der Einordnung in eine der Kategorien aus 3.1, wird LEO in die erste Kategorie „Lernen über Relationen" eingegliedert. Dies begründet sich damit, dass LEO etwas über die Statistiken lernt und damit etwas über Relationen. Der Versuch der Zuordnung des Algorithmus zu einer der Methoden des maschinellen Lernens ist nicht ganz einfach. Doch mit einem gewissen Abstand kann man die Art des Lernens mit dem eines Neuronalen Netzes vergleichen, wobei die Knoten die Statistiken mit den Funktionen zur Bestimmung der Selektivitäten sind. Die Anpassungen entsprechen den Gewichten an den Eingängen (Abb. 3.3).

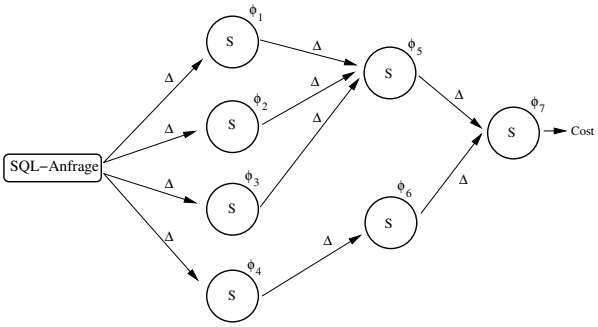

Abbildung 3.3: Neuronales Netz zu LEO

3.2.2 ST-Histograms Microsoft's Self Tuning Histograms

Im Projekt ST-HISTOGRAMS [AC99], geht es um sich selbst anpassende Histogramme. Mit ihnen sollen die Selektivitätsberechnungen des Anfrageoptimierers verbessert werden. Dabei werden die Histogramme einmal initialisiert und danach durch die Rückkopplung mit der Anfrageausführungseinheit immer weiter verfeinert. Die Verfeinerung findet auf zwei Ebenen statt. Einerseits werden die relativen Häufigkeiten der Klassen angepasst. Zum anderen werden nach bestimmten Zeiträumen die Histogramme mit ihren Klassen neu eingeteilt, also restrukturiert. Die Restrukturierung hat das Ziel, innerhalb der einzelnen Klassen eine möglichst gleichmäßige Verteilungen zu erhalten. Auf Bild 3.4 ist der schematische Ablauf zu sehen. Der Algorithmus 1 stellt den generellen Vorgang des ersten Teils, Anpassung der Histogramme, für den 1-dimensionalen Fall dar.

Bei der Einteilung ist, wie bei dem Projekt LEO, die Gruppe des Lernens über Relationen zu wählen, da auch hier Statistiken gelernt und verfeinert werden, und somit etwas über die Relationen selbst gelernt wird. Die Zuordnung zu einer Methode aus der KI ist, wie bei LEO, schwer. Aber auch hier ist der Vergleich mit einem Neuronalen Netz möglich. Jedoch sind die Knoten dann nur noch die Funktionen zur Bestimmung der Selektivitäten und die Gewichte der Eingänge würden dann die entsprechenden Histogramme ([Ioa03]) darstellen.

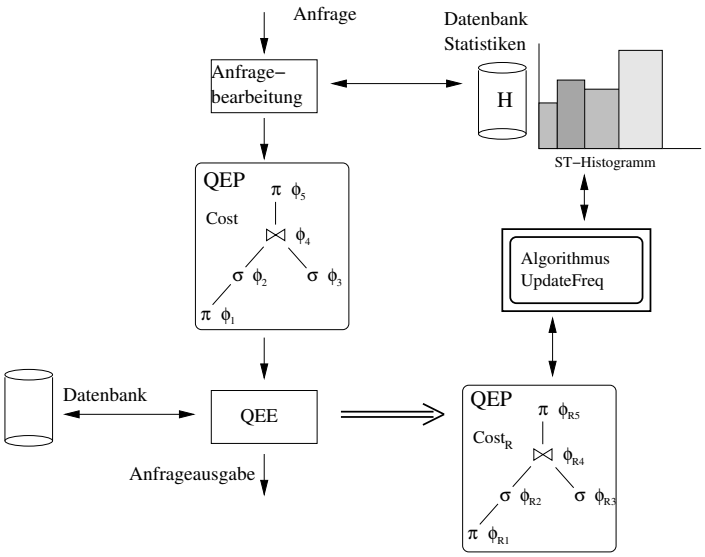

Abbildung 3.4: Anfragebearbeitung mit ST-Histograms

Algorithmus 1 Anpassung der relativen Häufigkeiten - UpdateFreq

Eingabe: Histogramm H und gemessene Selektivität act

Ausgabe: Histogramm H mit angepassten Klassenhäufigkeiten

begin

1. bestimme die k Klassen aus H, die mit der Bereichsanfrage überlappen, $\{b_1, b_2, ..., b_k\}$
2. berechne die geschätzte Selektivität est mit Hilfe des Histogramms H
3. berechne den Fehler $esterr$ zwischen geschätzter und gemessener Selektivität, $esterr = act - est$
4. Verteilung des Fehlers auf die k Klassen. Dabei tragen Klassen mit einer größeren relativen Häufigkeit mehr zum Fehler bei als Klassen mit einer kleineren relativen Häufigkeit

end

3.2.3 Vergleich von LEO und ST-Histograms

Bei dem Vergleich der beiden Ansätze sind zwei Hauptunterschiede festzustellen:

1. mit LEO können alle Selektivitätsberechnungen korrigiert werden, auch solche, die keine Histogramme zur Grundlage haben. Im Gegensatz können mit dem Projekt ST-HISTOGRAMS nur Berechnungen verbessert werden, die Histogramme zur Grundlage haben. Modellannahmen des Optimierers können somit nicht korrigiert werden.

2. der zweite Hauptunterschied besteht in der Nutzung oder Speicherung der berechneten Änderungen. MICROSOFT geht bei seinem Ansatz den Weg, die Änderungen immer direkt auf die Histogramme anzuwenden und somit nicht extra zu speichern. IBM dagegen speichert die berechneten Korrekturen getrennt von den Statistiken und verändert somit immer nur die Korrekturen selbst. Dieser Ansatz hat mehrere Vorteile, aber auch Nachteile.

Vorteile des LEO-Ansatzes	Nachteile des LEO-Ansatzes
• Möglichkeit der Speicherung von Anpassungen ohne die Existenz der entsprechenden Histogramme • einfache Abschaltung des Lernens ist möglich	• zusätzlicher Speicher wird benötigt • zusätzliche Berechnungen entstehen durch die Verknüpfung von Anpassungen und Statistiken

Tabelle 3.1: Vor- und Nachteile von LEO gegenüber von ST-Histograms

3.2.4 Automatische Statistikverwaltung für den Anfrageoptimierer

Das Projekt AUTOMATIC STATISTICS MANAGEMENT FOR QUERY OPTIMIZER soll den Prozess der Auswahl von Statistiken automatisieren [CN00]. Für eine gegebene Datenbank ist die Menge von möglichen Statistiken sehr groß. Da nicht alle Statistiken nutzbringend sind und anderseits auch aus Platzgründen nicht die Möglichkeit besteht, alle Statistiken zu erstellen, ist es wichtig, nützliche von weniger nützlichen Statistiken zu unterscheiden. Und genau in diesem Punkt setzt die Arbeit von „AUTOMATIC STATISTICS MANAGEMENT FOR QUERY OPTIMIZER" an. Dabei wird für eine gegebene Menge von Anfragen (engl. Workload) eine optimale und somit minimale Menge von Statistiken gesucht. Diese Menge soll es dann der Anfrageoptimierung ermöglichen, die Selektivitäten für den Workload möglichst genauso gut zu bestimmen, als für jede andere Menge von Statistiken. Zur Beschreibung des Algorithmus werden folgende Definitionen benötigt ([CN00]):

Kosten einer Anfrage Q bzgl. der Menge von Statistiken S, wird symbolisiert durch $Cost(Q_S)$ und entspricht dem Ergebnis der Kostenberechnung der Anfragebearbeitung von Q, bei der der Anfrageoptimierer nur die Menge S von Statistiken benutzt hat.

Kostengleichheit von zwei Mengen von Statistiken (S_1, S_2) bzgl. der Anfrage Q wird symbolisiert durch $Cost(Q_{S_1}) = Cost(Q_{S_2})$. Es werden die drei folgenden Kostengleichheiten definiert:

- **Ausführungsplan Gleichheit,** wenn $Cost(Q_{S_1}) = Cost(Q_{S_2})$ und die beiden erzeugten Anfrageausführungspläne gleich sind
- **Optimierer-Kosten Gleichheit,** wenn $Cost(Q_{S_1}) = Cost(Q_{S_2})$
- **t-Optimierer-Kosten Gleichheit,** wenn $Cost(Q_{S_1})$ und $Cost(Q_{S_2})$ um maximal t Prozent voneinander abweichen

Kandidaten-Menge von Statistiken bzgl. der Anfrage Q ist die Menge von Statistiken, die die Generierung des Anfrageausführungsplans

bei der Anfragebearbeitung von Q beeinflussen. Das Symbol für diese Menge ist im weiteren K.

Notwendige-Menge von Statistiken N bzgl. der Anfrage Q ist eine minimale Teilmenge aus der Kandidaten-Menge K. Bei dieser Teilmenge N soll für den Anfrageausführungsplan von Q die gleichen Kosten berechnet werden, als wenn die gesamte Menge K von Statistiken vorhanden gewesen wäre. Es soll also gelten:

- $N \subseteq K$ und
- $Cost(Q_N) = Cost(Q_K)$ und
- N ist minimal, d.h. $\neg\exists S \subseteq K \mid Cost(Q_S) = Cost(Q_K) \wedge |S| < |N|$.

Diese Definitionen werden wie folgt auf eine Menge von Anfragen Θ erweitert:

- die Kandidaten-Menge von Statistiken N_Θ bzgl. Θ ist die Vereinigung der Kandidaten-Mengen jeder einzelnen Anfrage in Θ
- die Notwendige-Menge von Statistiken bzgl. Θ ist die Teilmenge aus N_Θ, die kostengleich zu $N_\Theta$bzgl. jeder Anfrage in Θ ist und die Minimalität erfüllt.

Der Algorithmus 2 beschreibt den Ablauf zur Berechnung einer Notwendige-Menge von Statistiken.

Das Projekt der automatischen Statistikverwaltung für den Anfrageoptimierer sollte in die Gruppe „Lernen bezüglich der Anfragen, die an das DBMS gestellt werden" aus 3.1 eingeordnet werden, da mit dieser Arbeit etwas aus einer Menge von Anfragen oder einzelnen Anfragen abgeleitet wird.

3.2.5 Partitionierungsratgeber

Im zunehmendem Maße werden immer mehr Parallele Datenbanken eingesetzt, um höhere Leistungen in DBSen zu erreichen. Dabei ist eine Art

Algorithmus 2 Berechnung einer Notwendige-Menge von Statistiken - NotwendigeStatistiken

Eingabe:

- Menge von Anfragen, Θ

Ausgabe: Menge von Notwendigen Statistiken, N_Θ

foreach(Anfrage Q in Θ)

1. prüfe, ob N_Θ bereits eine Notwendige-Menge von Statistiken für Q ist. Wenn ja, nächste Anfrage Q aus Θ und gehe zu 1., sonst weiter mit 2.

   (a) Berechnung der Pläne P_{low} und P_{high} aus der Anfrage Q

   - bei der Berechnung P_{low} werden für alle Selektivitäten, die keine Statistiken zur Grundlage haben, ein Minimalwert ε genutzt. Bei P_{high} werden für diese Selektivitäten der Wert $1 - \varepsilon$ verwendet

   (b) Vergleich von $cost(P_{low})$ und $cost(P_{high})$ auf t-Optimierer-Kosten Gleichheit

2. suche Statistik s, die die Berechnung von $Cost(Q)$ positiv beeinflusst und setze $N_\Theta = N_\Theta \cup \{s\}$ und gehe zu 1.

endforeach

der Parallelisierung von DBSen, ein Netz von Rechnern zu haben. Diese teilen sich keinerlei Betriebsmittel und sind nur mittels eines Kommunikationsnetzwerkes verbunden. Diese Art von Verbindung wird im englischen mit dem Begriff *Shared-Nothing-Environment* bezeichnet. In diesem Netz arbeitet ein verteiltes DBS. Um einen hohen Grad der Parallelität zu ermöglichen, ist es notwendig, die gesamte Datenbank auf alle Knoten (Rechner) zu verteilen. Dabei werden die einzelnen Relationen horizontal zerteilt und auf die unterschiedlichen Knoten abgelegt. Diese Zerteilung nennt man Partitionierung und entsteht mit Hilfe von Partitionsfunktionen über einer Teilmenge der Attribute der Relation. Diese Teilmenge wird als Partitionsschlüssel bezeichnet. Die Bestimmung der optimalen Partitionierung ist ein großes Entwurfsproblem. Das Projekt „AUTOMATING PHYSICAL DATABASE DESIGN IN A PARALLEL DATABASE" versucht diesen komplizierten Entwurfsprozess zu automatisieren [RZML02].

In [RZML02] wird eine Methode beschrieben, die die horizontale Aufteilung von Relationen zwischen den einzelnen Knoten eines Shared-Nothing Systems in Bezug zu einer Menge von Anfragen optimiert. Dabei wird auf das Kostenmodell des Anfrageoptimierers zurückgegriffen, um die Güte einer Partitionierung zu berechnen. Der Algorithmus 3 zeigt den generellen Ablauf zur Berechnung einer optimalen oder nahezu optimalen Partitionierung. Als Eingabe dient eine Menge von Anfragen und ihre Häufigkeit. Daraufhin wird für jede Relation jeder Anfrage die beste Partitionierung gesucht. Die gesamte Menge von Partitionierungen stellt dann die Menge von Kandidat-Partitionen dar. Dabei kann es durchaus vorkommen, dass für eine Relation zwei oder mehr Kandidaten existieren. Im Schritt 2 werden dann alle Kandidaten bzgl. des gesamten Workloads und den entsprechenden Häufigkeiten bewertet. Der Schritt 3 gibt dann die besten Kandidaten aus. Zu den Einzelheiten des Algorithmus siehe [RZML02].

Die Einordnung des Projektes „AUTOMATING PHYSICAL DATABASE DESIGN IN A PARALLEL DATABASE" in eine der vier Kategorien aus 3.1 ist nicht ganz einfach und eindeutig. Das Ziel dieses Projektes ist es, etwas über die physische Datenstruktur zu lernen. Daher sollte die Grup-

Algorithmus 3 Berechnung einer optimalen Partitionierung - Optimale-Partitionierung

Eingabe: Menge von Anfragen Θ und ihre Häufigkeiten

Ausgabe: eine möglichst optimale Partitionierung der Datenbank

1. berechne für jede Anfrage $Q \in \Theta$ eine Menge von Kandidat-Partitionen
   - dabei wird für jede Relation in Q die beste Partitionierung bzgl. Q erstellt
2. bewerte die Kandidat-Partitionen mit Hilfe des Anfrageoptimierers bzgl. der gesamten Menge von Anfragen Θund ihren Häufigkeiten
3. Ausgabe der besten Partition pro Relation aus der Menge der Kandidat-Partitionen

pe „Lernen über Relationen und die physische Datenstruktur der Datenbank" gewählt werden. Aber diese physische Struktur oder Aufteilung ist immer an die angegebene Menge von Anfragen geknüpft und nur für diese wird die berechnete Partitionierung optimal sein. Daher wird auch aus Anfragen gelernt. Dies hätte die Wahl der Gruppe „Lernen bezüglich der Anfragen, die an das DBMS gestellt werden" zur Folge. Somit sollte das Projekt in beide Gruppen eingeordnet werden und ist ein Beispiel für Überschneidungen der einzelnen Gruppen.

3.2.6 Indexberater

Die letzten Versionen von kommerziellen relationalen DBMSen haben nahezu alle ein Werkzeug, das bei der Erstellung von Indexen Unterstützung gibt. Dabei ist immer die Frage, von welchen Indexen würde eine Menge von Anfragen am meisten profitieren. Diese Werkzeuge benötigen daher als Eingabe einen Workload, also eine Menge von Anfragen, die relativ häufig gestellt werden. Daraus werden dann zum einen Vorschläge für neu aufzubauende Indexe berechnet, zum anderen werden bereits im

System vorhandene Indexe auf ihren Nutzen geprüft. Das Ergebnis dieser Analyse wird dann beim Nutzer in entsprechender Form dargestellt. Zur Analyse wird in den Systemen von MICROSOFT, SQL-SERVER, und IBM, DB2, das Verfahren „WHAT IF INDEX" zur Indexsimulation verwendet [CN97, CN98a, CN98b, VZZ+00, LL02]. Dieses Verfahren soll im folgenden vorgestellt werden.

„What If Index" - Indexanalyse und Indexsimulation

Mit dem Verfahren „WHAT IF INDEX" soll eine Menge von Indexen bezüglich einer Menge von Anfragen auf ihren Wert geprüft werden. Das Verfahren bietet dabei die Möglichkeit, Indexe zu bewerten, die noch nicht im System existieren. Außerdem wird die Möglichkeit gegeben, diese Bewertungen nicht nur gegen die aktuelle Größe der Datenbank zu berechnen, sondern auch gegen mögliche Skalierungen der Datenbank. Um bei der Bewertung eines Indexes auch mit dem Kostenmodell des Optimierers übereinzustimmen, wird der Anfrageoptimierer selbst für diese Bewertung genutzt. Die Übereinstimmung der Kosten des Optimierers mit der Bewertung des Verfahrens ist grundlegend, da bei der späteren Anfragebearbeitung dieses Kostenmodells über die Nutzung eines Indexes entschieden wird. Die Nutzung des Anfrageoptimierers wirft in der Praxis einige Probleme auf [CN98a]. Ein Problem ist, dass nicht vorhandene Indexe in Betracht gezogen werden sollen, ohne sie vorher komplett aufbauen zu müssen. Die Möglichkeit der Skalierung der Datenbank stellt ein anderes Problem dar. Diese Skalierung wird mit Hilfe der Systemstatistiken dargestellt. Eine Änderung dieser Statistiken könnte aber bei der gleichzeitigen Anfragebearbeitung realer Anfragen zu falschen Ausführungsplänen führen. Dadurch würde die Abarbeitung realer Anfragen negativ beeinflusst werden. Zur Beschreibung des Algorithmus benötigen wir zuerst noch einige Definitionen.

hypothetischer Index ist ein Index, der nur durch seine Definition und durch Statistiken für ihn existiert. Der Index selbst hat keine Extension, d.h. er kann nur bei der Anfrageoptimierung genutzt werden

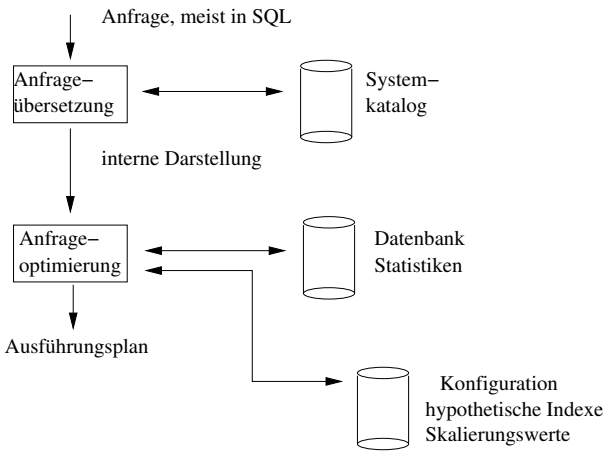

Abbildung 3.5: Indexsimulation

aber nicht bei der Anfrageausführung.

Konfiguration ist eine virtuelle Sicht auf das DBS. Eine Konfiguration besteht aus einer Menge von Indexen und aus Skalierungswerten. Die Indexe können bereits im System bestehen oder hypothetische Indexe sein. Die Skalierungswerte sollen es ermöglichen, die Datenbank virtuell zu vergrößern oder zu verkleinern. Bei der Optimierung werden sie mit den entsprechenden Datenbankstatistiken multipliziert.

Auf dem Bild 3.5 ist der schematische Ablauf der Anfragebewertung mit Hilfe von „WHAT IF INDEX" skizziert. Dabei werden dem Verfahren eine Konfiguration und eine Menge von Anfragen übergeben. Bei der Optimierung wird wie bei der normalen Optimierung vorgegangen, nur dass in diesem Fall allein die Indexe aus der Konfiguration berücksichtigt werden. Die Statistiken werden mit Hilfe der Skalierungswerte für die Berechnungen verändert. Am Ende des Verfahrens werden die erzeugten Pläne auf die Nutzung der Indexe untersucht und daraus eine Bewertung berechnet.

Auch das Projekt „WHAT IF INDEX" zur Indexanalyse und Indexsimulation wollen wir in eine der vier Kategorien aus 3.1 einordnen. Das Ziel

dieses Projektes ist es, etwas über Indexe und ihren Wert bzgl. einer Menge von Anfragen zu lernen. Das zu Lernende wird also aus Anfragemustern abgeleitet. Damit würde die Gruppe „Lernen bezüglich der Anfragen, die an das DBMS gestellt werden" die richtige sein. Nun sind aber Indexe Hilfsstrukturen für Relationen und somit trifft auch die Gruppe „Lernen über Relationen und die physische Datenstruktur der Datenbank" zu. Somit ist auch dieses Projekt ein Beispiel für die Zusammenhänge der einzelnen Gruppen.

3.3 Lernen in AQuES

In diesem Abschnitt soll zuerst AQuES ([Sti00]) kurz vorgestellt werden. Danach sollen die Möglichkeiten des Lernens im System von AQuES dargestellt werden.

3.3.1 AQuES - ein Agentenbasiertes Anfrageausführungssystem

AQuES steht für Agent-based Query Evaluation System, es ist also ein Anfrageausführungssystem, das mit Hilfe von Konzepten aus dem Bereich der Software-Agenten realisiert wurde. Der Grundgedanke von AQuES ist es, ein System aus kooperierenden verteilten Komponenten aufzubauen, um so einen flexiblen Ansatz zur Anfrageoptimierung und Auswertung zu ermöglichen. Ein AQuES-System besteht dabei aus einer Menge von AQuES-Agenten. Ein AQuES-Agent ist eine komplexe Funktionseinheit von AQuES und wird aus einer Menge von Komponenten gebildet. Eine Komponente ist eine kleinste funktionale Untereinheit des gesamten Systems. An die Architektur von AQuES wurden verschiedenste Anforderungen gestellt, darunter befinden sich auch die Möglichkeiten der dynamischen Optimierung und des Lernens. Der dynamische Optimierer und der Lerner bilden jeweils eine Komponente von AQuES. [Sti00, SSF01]

3.3.2 Der Lerner und dynamische Optimierer in AQuES

Die Komponenten des Lerners und des dynamischen Optimierers stehen für das AQuES-System die Möglichkeiten des Lernens zur Verfügung. Der Lerner soll Daten und Statistiken über alle Betriebsmittel (Daten, Hardware, andere Agenten) sammeln und eigene Ausführungsparameter messen und weitergeben. Diese Daten sollen auch dazu dienen, die Optimierung und die Ausführungsstrategie zu verbessern. Der dynamische Optimierer soll es ermöglichen, bereits erzeugte und gestartete Anfrageausführungspläne zu unterbrechen und Teile des Plans neu zu berechnen und zu optimieren. Diese Unterbrechung soll anhand der Daten, die von der Lerner-Komponente kommen, gesteuert werden. Weiter könnte der Lerner neben den anderen Gebieten auf dem Gebiet „Lernen über die Topologie eines verteilten DBSs" tätig sein. Dabei wären sicherlich folgende Punkte von Bedeutung:

- Welche Agenten sind aktiv und wo sind diese?
- Sollten zusätzliche Agenten gestartet oder gestartete Agenten gestoppt werden ?
- Wäre es sinnvoll Agenten im System zu verschieben?

Kapitel 4

Lernen durch Verwendung von Statistiken

In diesem Kapitel soll die Verwendung von Statistiken zur Leistungssteigerung in DBSen in den Vordergrund gestellt werden. Wie bereits im Abschnitt 2.2 - „Anfragebearbeitung in DBMSen" vorgestellt, spielen Statistiken eine große Rolle bei der Anfrageoptimierung. Im speziellen sollen dabei Histogramme betrachtet werden. Dazu wird im ersten Abschnitt die Entwicklung der Histogramme in DBMSen vorgestellt. Im zweiten Abschnitt liegt der Fokus auf multidimensionale Statistiken, ihren Nutzen und ihre Probleme. Im letzten Abschnitt dieses Kapitels wir mein Konzept für multidimensionale Statistiken vorgestellt.

4.1 Entwicklung der Histogramme in DBMSen

Histogramme werden schon lange in Bereichen benutzt, in denen Statistiken eine große Rolle spielen, wie z.B. in der Finanzwirtschaft, Medizin und der Technischen Informatik. Ihre graphische Darstellung wurde vom Anfang bis heute zur Visualisierung von Statistiken genutzt. In DBMSen wird allerdings vielmehr auf das mathematische Gerüst von Histogrammen zurückgegriffen. Seit den frühen Achtzigern werden Histogramme zur Approximierung von Datenverteilungen in DBMSen eingesetzt. An-

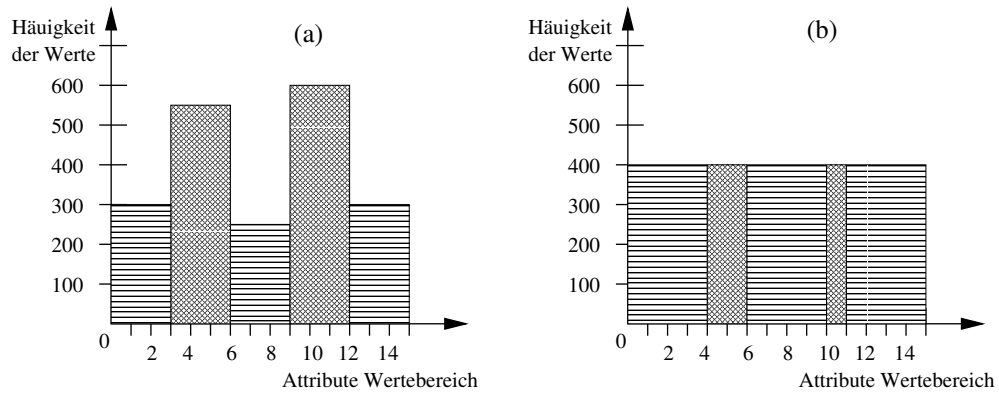

Abbildung 4.1: Histogrammtypen

gewendet werden die Histogramme in DBMSen hauptsächlich für die Selektivitätsabschätzungen von Anfragen bei der Anfragebearbeitung und zur approximativen Beantwortung von Anfragen. Eine ausführliche Definition eines Histogramms gibt IOANNIDIS in [Ioa03]. Allgemein kann man den Aufbau eines Histogramms H für ein Attribute X der Relation R so beschreiben:

- die Datenverteilung von X wird in $\beta\,(1 \leq \beta)$ Klassen partitioniert
- dann wird die Häufigkeit jeder Klasse bestimmt

Bei der Klasseneinteilung besteht ein großer Freiraum, woraus die unterschiedlichen Arten von Histogrammen hervorgehen. Die ersten Histogramme, die in DBMSen genutzt wurden, waren *Equi-width* Histogramme (Bild 4.1(a)). Bei *Equi-width* Histogrammen wird der Wertebereich des Attributs in sich nicht überschneidenden Klassen gleicher Größe eingeteilt. Weiterentwicklungen waren *Equi-depth*, *Equi-high* und *Equi-sum* Histogramme (Bild 4.1(b)). Bei *Equi-depth*, *Equi-high* und *Equi-sum* Histogrammen werden die Klassen so eingeteilt, dass sich für allen Klassen die gleiche Häufigkeit ergibt.

Danach wurden noch weitere Histogrammtypen entwickelt, wie *V-optimal*, *MaxDiff* und *Compressed* (näheres siehe [Ioa03]).

In den kommerziellen DBMSen von heute wird meist einer der zuvor vorgestellten 1-dimensionalen Histogrammtypen verwendet.

4.1.1 Multidimensionale Histogramme

Neben den 1-dimensionalen Histogrammen wurde auch an Konzepten für mehr-dimensionalen Histogramme geforscht. Dabei ist das größte Problem, dass viele Konzepte bei höher dimensionalen (>3) nicht mehr skalieren und in ihren Datenmengen auswuchern und somit deren Anwendung in kommerziellen Datenbanksystemen nicht möglich ist. Ein anderes Problem ist die Identifizierung der notwendigen Attribute einer Relation für ein mehr-dimensionales Histogramm. Diese Aufgabe wurde laut [Ioa03] als ein NP-hartes Problem identifiziert. Was bedeutet, dass die komplette Lösung eine exponentielle Laufzeit hat. Die Anwendung von mehr-dimensionalen Histogrammen ist, wie bei 1-dimensionalen Histogrammen, die Bestimmung von Selektivitäten für die Anfrageoptimierung und die approximative Beantwortung von Anfragen. Der Aufbau von mehr-dimensionalen Histogrammen ist dem für 1-dimensionale sehr ähnlich. Zuerst wird dabei der Attributwerteraum in Teilräume zerlegt und dann für diese Teilräume die Kardinalität bestimmt. Weitergehende Informationen zu mehr-dimensionalen Histogrammtypen, ihr Aufbau und ihre Benutzung, werden in [Ioa03, WVI97, LKC99, JKM+98, JKNS99, KW99, FM99] gegeben.

4.1.2 Statische versus dynamische Histogramme

Ein anderer Aspekt ist die Art des Aufbaus und die Wartung von Histogrammen. Dieser Gesichtspunkt steht orthogonal zum Problem der Dimensionalität des Histogramms. Es existieren dabei die Möglichkeiten von statischen sowie dynamischen, adaptiven Histogrammen. Statische Histogramme werden einmal komplett aufgebaut und danach nicht mehr verändert, bis der Nutzer oder das System den kompletten Aufbau neu bestimmt. Somit können statische Histogramme nur die Datenverteilung

zur Zeit des Aufbaus widerspiegeln. Dynamische, adaptive Histogramme passen sich stetig an die Datenverteilung an und können somit die aktuelle Verteilung darstellen. Der Nachteil von dynamischen Ansätzen ist der zusätzliche Aufwand, um einerseits die entsprechenden Änderungen in der Datenverteilung festzustellen und andererseits dann das Histogramm entsprechend anzugleichen. Somit entsteht hier ein „Trade-off", also eine Situation, in der zwischen Aktualität und zusätzlichen Aufwand gewählt werden muss.

4.2 Lernen von Statistiken durch Rückkopplung

Im folgenden Abschnitt soll ein Konzept zum Lernen von Statistiken durch Rückkopplung vorgestellt werden, das im Rahmen dieser Diplomarbeit entwickelt wurde.

4.2.1 Konzept zum Lernen von Statistiken

Statistiken und ihre Genauigkeit sind von enormer Bedeutung für die Anfrageoptimierung, da sie die Grundlage für die Modellberechnungen der Optimierung darstellen (siehe Abschnitt 2.2). Der Aufbau und die Pflege speziell von mehr-dimensionalen Statistiken ist ein teurer und aufwendiger Prozess. Unser Konzept soll den Aufbau und die Pflege von solchen Statistiken automatisieren und somit praktikabel machen. Bei der Ausarbeitung unseres Konzeptes stand dessen Nutzbarkeit für Datenbanksysteme im Vordergrund. Aus diesem Punkt ergeben sich folgende Bedingungen und Möglichkeiten:

1. für jede Relation sollten so wenig Statistiken wie möglich existieren. Aber diese existierenden Statistiken müssen genügend Aussagen zulassen, um eine bestmögliche Anfrageoptimierung zu erlangen. Am besten wäre also eine n-dimensionale Statistik, die auch über alle enthaltenen Teilmengen von Attributen statistische Aussagen erlaubt. Dieses ist aber wie auf Bild 4.2 zu erkennen, nicht immer gegeben.

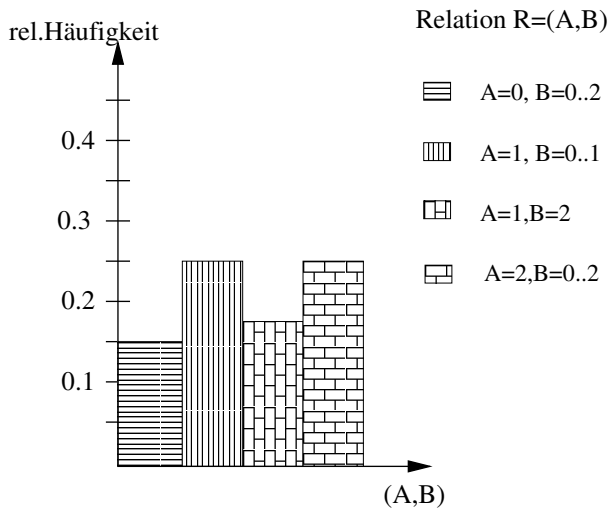

Abbildung 4.2: 2-dimensionales Histogramm

Mit Hilfe des Histogramm im Beispiel ist es möglich, Aussagen über die Häufigkeit von Paaren bestehend aus A und B zu treffen. Es ist auch die Verteilung der Werte des Attributs A abzulesen. Allerdings kann über die Verteilung von B kaum eine Aussage getroffen werden.

2. die Statistiken bzw. die Daten der Statistiken müssen in einer Struktur gespeichert werden, die nicht nur eine Hauptspeicherstruktur darstellt, sondern auch auf den Sekundärspeicher ausgelagert werden kann.

3. ein möglichst schneller und effizienter Zugriff auf die Daten muss gewährleistet sein.

4. die Nutzung von Aussagen oder Ergebnissen der Anfrageausführung soll durch deren Rückkopplung ermöglicht werden.

Im Hinblick auf die Art des Aufbaus der Statistiken, ergeben sich folgende zwei Möglichkeiten:

vollständig-adaptierbar die Statistik wird zu Beginn einmal komplett

aufgebaut und wird sich dann der sich ändernden Verteilung anpassen.

spärlich-adaptierbar die Statistik wird punktweise durch die Anfragen selbst aufgebaut und passt sich dann ebenfalls der Verteilung an.

Der Vergleich der beiden Möglichkeiten liefert folgende Ergebnisse:

vollständig-adaptierbar	spärlich-adaptierbar
• bereits die erste Anfrage kann mit Hilfe der Statistik optimiert werden. • es muss jeder berechnete Wert der Statistik gespeichert werden, ohne Hinblick auf dessen Nutzen.	• um die Statistik nutzen zu können, muss erst eine gewisse Trainingsphase durchlaufen worden sein. • die Statistik ist speziell an Orten häufiger Anfragen sehr genau. • an Orten ohne Anfragen sind keine Werte vorhanden und müssen somit auch nicht gespeichert werden. Allerdings können diese Orte i.A. auch nicht für die Anfrageoptimierung genutzt werden.

Tabelle 4.1: Vergleich der Arten des Aufbaus der Statistik

Im Sinne der Anpassung an der aktuellen Datenverteilung verhalten sich beide Arten des Aufbaus gleich. Sie werden sich hauptsächlich in den Regionen mit häufig auftretenden Anfragen der Verteilung anpassen. In meinem Konzept habe ich mich für die spärlich-adaptive Variante entschieden. Diese Variante kommt dem Vorgang des Lernens deutlich näher. Der Hauptvorteil liegt aber darin, dass nur Werte abgespeichert werden,

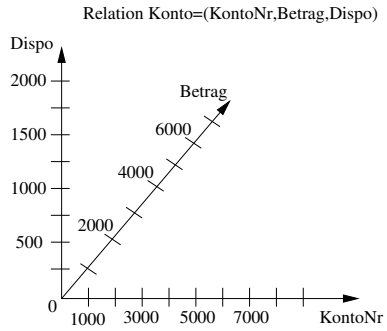

Abbildung 4.3: Darstellung Werteraum

die auch wirklich schon einmal gebraucht wurden, und somit mit gewisser Wahrscheinlichkeit wieder benötigt werden.

Bevor ich zu weiteren Beobachtungen komme, soll noch eine Definition gegeben werden:

Werteraum der Werteraum für eine n-dimensionale Statistik ist ein n-dimensionaler Raum, wobei jede Dimension durch ein Attribut und dessen Wertevorrat dargestellt wird. Der Werteraum ist somit immer von der Extension der zugrunde liegenden Relation abhängig. Er gibt keine Auskunft über die Häufigkeit des Attributwertes. Auf Bild 4.3 ist ein Beispiel dargestellt, dabei würde der Werteraum der Quader sein, der alle momentanen Werte der Attribute enthält.

Bei der Ausarbeitung des Entwurfs wurden noch folgende wichtige Beobachtungen gemacht:

1. um eine möglichst genaue Verteilung darstellen zu können, muss die Klassenaufteilung des Werteraumes möglichst viele und kleine Klassen haben. Die Begründung dafür ist die Gleichverteilungsannahme innerhalb einer Klasse.

2. es sollte die Möglichkeit geben, zwischen gemessenen und berechneten Statistikwerten zu unterscheiden. Dabei sind gemessene Werte, die Messwerte, die unverändert direkt aus der Anfrageausfüh-

rungseinheit in die Statistik übernommen werden können. Berechnete Werte stellen Selektivitäten dar, bei denen die Messwerte durch Algorithmen zur Anpassung auf die vorhandene Statistik verändert wurden, und diesen Algorithmen bestimmte Annahmen über die Verteilung zugrunde liegen.

3. um möglichst viele gemessene Werte direkt abspeichern zu können, ist es notwendig, Klassen des Werteraumes wieder zusammenzufassen und als eine Klasse abzuspeichern zu können. Dabei sollen Statistiken der enthaltenen Klassen möglichst erhalten bleiben, speziell sollten gemessene Statistikwerte nicht verloren gehen. Das Resultat ist somit eine Statistik mit unterschiedlichen Klassengrößen.

4. es wird eine eindeutige Abbildung der Prädikate einer Anfrage auf den Wertebereich benötigt. Dabei sollen immer Klassen maximaler Größe benutzt werden. Das soll einerseits den Rechenaufwand klein halten und andererseits eine möglichst genaue Abschätzung der Selektivität für die Anfrage ermöglichen.

Bei der Entwicklung des Entwurfs mussten zwei Einschränkungen getroffen werden:

1. die Einschränkung auf 2-dimensionale Statistiken, da ein Konzept für den n-dimensionalen Fall die Komplexität der Algorithmen stark erhöht und somit den Rahmen dieser Diplomarbeit übersteigt.

2. eine andere Einschränkung betrifft die Möglichkeit, dass Anfragen nicht genau auf die Klassengrenzen treffen. Daraus müsste sich eine Skalierung der gemessenen Werte auf die jeweiligen Klassengrenzen ergeben. Auch dieses erhöht die Komplexität der Algorithmen in einem Maße, die nicht im Rahmen dieser Diplomarbeit bearbeitet werden kann. Daher sollen die Anfragen so beschränkt werden, dass sie immer auf Klassengrenzen der kleinsten Klassen treffen. Somit kann jede Anfrage auf eine Vielzahl von kleinsten Klassen abgebildet werden.

Im folgenden soll der konzeptuelle Ablauf dargestellt werden. Dabei sollen der Schwerpunkt speziell auf dem generellen Ablauf liegen. Die konzeptuellen Aspekte der einzelnen Algorithmen und Datenstrukturen werden dann im Kapitel 5 beschrieben. Im Kapitel 6 wird dann die Umsetzung des Konzeptes beschrieben. Dabei wird speziell die Partitionsfunktion, die ein Z-Ordering auf n-Ebenen ist, vorgestellt. Denn die Partitionsfunktion ist grundlegend für unser Konzept und ermöglicht die Anforderungen bezüglich der Sekundärspeicherstruktur zu erfüllen.

Der konzeptuelle Ablauf

Bevor wir den Ablauf der Selektivitätsberechnung und -anpassung beschreiben können, benötigen wir für jede Statistik eine Partitionsfunktion $P = P(A_1, A_2)$.

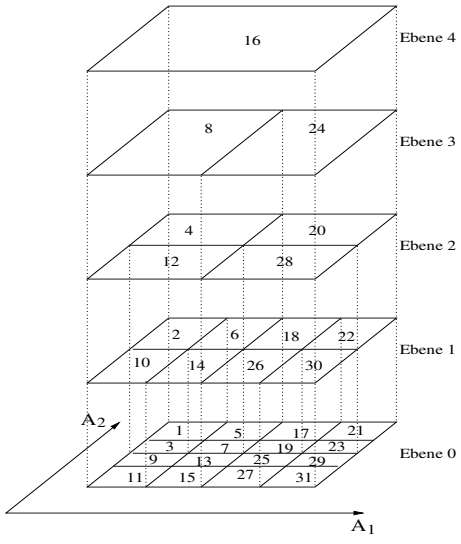

Abbildung 4.4: Klassenbildung

Diese Partitionsfunktion soll den Werteraum der Attribute A_1 und A_2 in Klassen aufteilen, und den einzelnen Klassen eindeutige Bezeichner geben. Auf Bild 4.4 ist eine mögliche Aufteilung des Werteraumes dargestellt. Dabei sind Klassen auf gleicher Ebene gleich groß. Je zwei nebeneinander liegende Klassen auf der Ebene e ergeben eine Klasse auf der Ebene $e + 1$. Die Zahlen innerhalb der einzelnen Klassen sollen die eindeutigen Bezeichner darstellen. Weiter wird eine Abbildungsfunktion $Abb = Abb(min_Q(A_1), min_Q(A_2), max_Q(A_1), max_Q(A_2))$ benötigt. Die Funktion Abb bildet die Prädikate der Attribute A_1und A_2 aus der Anfrage Q auf die einzelnen, maximalen Klassen der Partitionsfunktion P ab. Die Anfrage Q, $select * from R where 4 < A_1 < 15 \; and \; 1 < A_2$, würde dann mit der Abbildungsfunktion $Abb(5, 2,$

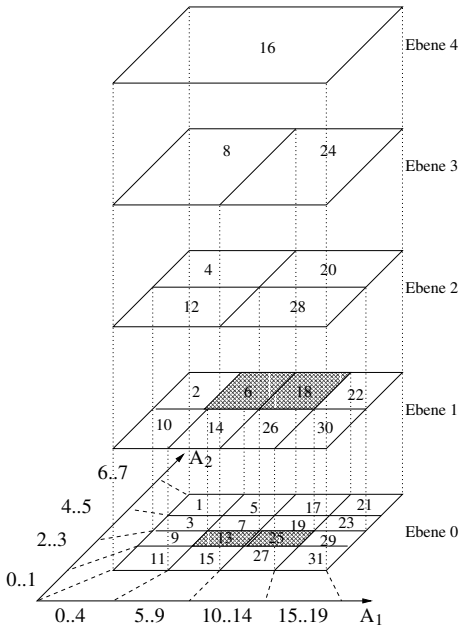

Abbildung 4.5: Abbildungsfunktion Abb

$14, *$) auf die Klassen 13, 25, 6 und 18 abgebildet werden (Abb. 4.5). Dabei wird der $*$ Operator als maximaler bzw. minimaler Wert im Werteraum des jeweiligen Attributs interpretiert. In unserem Beispiel würde $*$ somit als 7 aufgelöst oder interpretiert.

Als letztes werden noch zwei Funktionen benötigt, die das Auslesen von Selektivitäten aus der Statistik und das Einfügen von Werten aus der Anfrageausführung in die Statistiken erlaubt. Zunächst soll die Funktion zum Auslesen der Selektivität betrachtet werden. Diese Funktion soll mit $BerechneSel$ bezeichnet werden. Ihr wird eine Statistik S und eine Menge von Bezeichnern übergeben, die den selektierten Bereich im Werteraum darstellen. Daraufhin berechnet die Funktion $BerechneSel$ die Selektivität aus der Statistik und gibt diese zurück. Dabei wird auf die gemessenen bzw. berechneten Selektivitäten der einzelnen Klassen zurückgegriffen, in sofern diese vorhanden sind. Falls für eine Klasse noch keine Werte vorhanden sind, wird versucht mit Hilfe von Ableitungen aus der Statistik S, die Selektivität zu ermitteln, oder sie wird mit Hilfe von statisti-

schen Annahmen, wie Gleichverteilung und Attributsunabhängigkeitsannahme, berechnet. Die Funktion zur Anpassung der Statistik oder zum Einfügen von statistischen Werten aus der Anfrageausführung, soll im folgenden skizziert werden. Diese Funktion wird als $AnwendungSel$ bezeichnet. Dieser Funktion soll wie bei $BerechneSel$ eine Statistik und der selektierte Bereich übergeben werden. Dazu kommt noch der reale Wert der Selektivität, den die Anfrageausführungseinheit ermittelt hat. Dieser Wert wird dann auf die einzelnen Klassen aufgeteilt und in der Statistik abgespeichert. Bei der Aufteilung der Selektivität auf die einzelnen Klassen gibt es mehrere Möglichkeiten:

- eine Gleichverteilung der Selektivität Sel auf die Klassen der Anfrage. D.h. wenn n maximale Klassen im Anfragebereich sind, soll jede dieser Klassen die Selektivität $\frac{Sel}{n}$ erhalten.
- eine Gleichverteilung der Selektivität Sel auf die Klassen der Anfrage, mit Bezug auf die Klassengröße. Dabei hat eine Klasse k der Ebene 0, gerade die Größe 1, Klassen der Ebenen $e > 0$ haben eine Größe, die der Anzahl von enthaltenen Klassen der Ebene 0 entspricht. Im Falle der von uns gewählten Partitionsfunktion ist die Größe der Klasse auf der Ebene e gleich 2^e. Diese Größe der Klasse k wollen wir als $FE(k)$[1] bezeichnen. Dann folgt für die Verteilung der Selektivität Sel auf die n - Klassen $\{k_0, k_1, ..., k_n\}$ der Anfrage:

$$Selek(k_i) = \frac{Sel * FE(k_i)}{\sum_{m=0}^{n-1} FE(k_m)}$$

- bei bereits existierenden Klassen, kann der Fehler gleichmäßig auf die Klassen verteilt werden oder wie im Ansatz von ST-HISTOGRAM (siehe 3.2.2) den Fehler entsprechend der Selektivität der einzelnen Klassen aufteilen.

In meinem Konzept will ich eine Mischform aus gleichmäßiger Verteilung der Selektivität auf die Klassen unter Beachtung ihrer Größe und der

[1] FE - Flächeneinheiten

Gleichverteilung des Fehlers auf die einzelnen Klassen verwenden. Die Details dazu werden im Abschnitt 5.2 beschrieben.

Nun kann der Ablauf der Anfragebearbeitung, -optimierung und -ausführung, unter dem Gesichtspunkt unseres Konzepts dargestellt werden. Auf Bild 4.6 sind die einzelnen Abschnitte im Prozess der Anfrageauswertung abgebildet. Dabei werden als erstes aus der Anfrage Q mit Hilfe der Funktion Abb die Klassen der Statistik bestimmt, die durch die Anfrage ausgewählt wurden. Daraufhin werden bei der Anfrageoptimierung, diese entsprechenden Klassen und ihre Statistik benutzt, um unter Benutzung der Funktion $BerechneSel$ die Selektivität für diesen Bereich der Anfrage zu bestimmen. Als Abschluss werden die während der Anfrageausführung gemessenen Werte an die Funktion $AnwendungSel$ übergeben und somit entsprechend in die Statistiken eingearbeitet. Am Ende einer solchen Anfrageauswertung sind dann die beteiligten Statistiken an den Stellen der Anfrage neu aufgebaut, erweitert oder korrigiert worden. Somit kann das Wissen über die Selektivitäten aus der vergangenen Anfrage auf zukünftige Anfragen angewandt werden. Daher wird mit jeder Anfrage ein Lernprozess zu einer korrekteren Selektivitätsabschätzung vollführt.

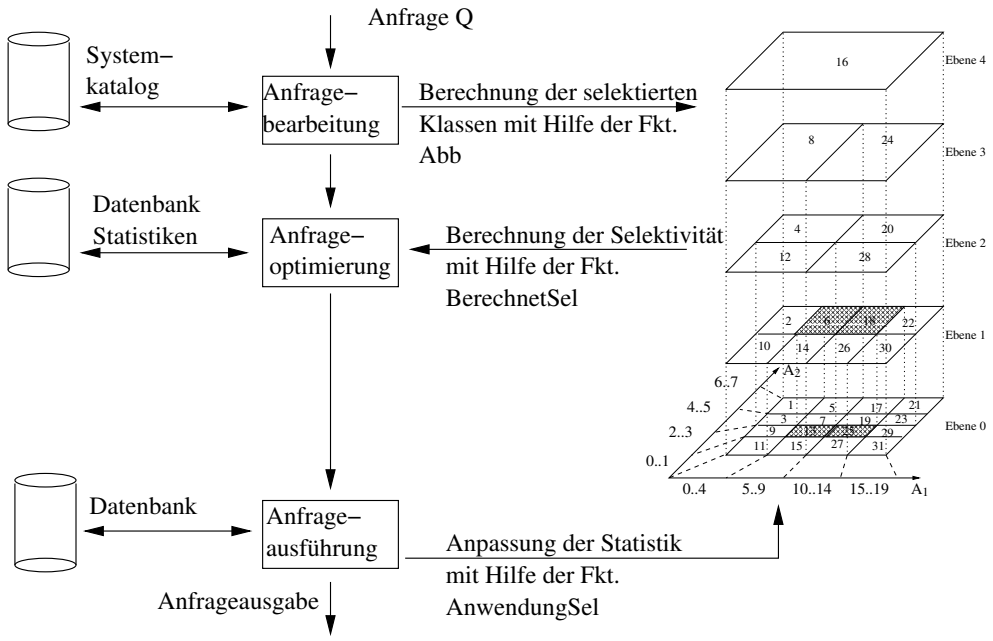

Abbildung 4.6: Ablauf der Anfrageauswertung

Kapitel 5

Algorithmen und Datenstrukturen

In diesem Kapitel sollen Algorithmen und Datenstrukturen vorgestellt werden, mit denen mein Konzept zum Lernen von Statistiken durch Rückkopplung umgesetzt werden kann. Dabei sollen speziell die beiden Funktionen $BerechneSel$ und $AnwendungSel$ näher vorgestellt werden. Um die Algorithmen der beiden Funktionen vorzustellen, ist es notwendig, eine Datenstruktur für die Klassen zu definieren. Diese Definition soll im ersten Abschnitt gegeben werden. Im Anschluss daran werden dann die Algorithmen vorgestellt. Dabei stehen in diesem Kapitel speziell das Konzept und die Funktionsweise im Vordergrund. Die Datenstrukturen und Algorithmen sollen daher möglichst implemenationsunabhängig vorgestellt werden. Da die beiden Funktionen zur Partitionierung P und zur Abbildung Abb sehr implementationsabhängig sind, werden sie als gegeben angesehen und erst im Kapitel 6 vorgestellt. Ebenso enthalten die Datenstrukturen keine implementationsspezifischen Details, sondern sollen möglichst abstrakt dargestellt werden.

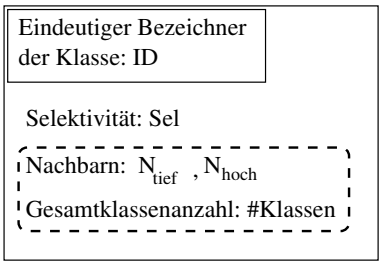

Abbildung 5.1: Datenstruktur

5.1 Datenstrukturen

In diesem Abschnitt soll die Datenstruktur der einzelnen Klassen einer Statistik vorgestellt werden. Wie bereits im Abschnitt 4.2 erwähnt wurde, soll die Möglichkeit bestehen, zwischen gemessenen und berechneten Werten zu unterscheiden. Ebenfalls soll es möglich sein, eine Menge von Klassen als ein Gebiet abzuspeichern. Um dieses zu ermöglichen, ist es notwendig, diese Menge von Klassen in einer entsprechenden Struktur abzuspeichern. Dazu wird in die Datenstruktur jeder Klasse die Möglichkeit eingebaut, zwei Nachbarn abzuspeichern, so dass die nächst tiefere und höhere Klasse abgespeichert werden kann. Tiefer und höher beziehen sich hierbei auf den Klassenbezeichner und somit auf die Ordnung der natürlichen Zahlen. Auf Bild 5.1 ist die Datenstruktur einer Klasse dargestellt. Dabei wird der eindeutige Bezeichner jede Klasse in der Statistik unter ID, gemessene bzw. berechnete Selektivität in Sel, und die möglichen Nachbarn unter N_{tief} und N_{hoch} gespeichert. Dabei soll die Klasse auch das Wissen über die Art der aktuell gespeicherten Selektivität haben. Die Felder für die Nachbarn können nur genutzt werden, wenn es sich um einen berechneten Wert handelt. Nicht gesetzte Nachbarn haben den Wert $NULL$. Um bestimmte Berechnungen zu vereinfachen, wird in $\#Klassen$ die Anzahl der Klassen gespeichert, die mit Hilfe der Nachbarn rekursiv erreicht werden können, plus die eigene Klasse.

Bezüglich der Datenstruktur sollen noch folgende Methoden definiert sein:

istVorhanden(Statistik S,KlassenId ID) diese Methode soll wahr zurückgeben, falls in der Statistik S, die Klassen mit dem Klassenbezeichner ID als gemessener oder berechneter Wert vorliegt, sonst ist der Rückgabewert falsch.

istGemessen(Statistik S,KlassenId ID) falls es sich um eine gemessene Selektivität in der Klasse ID in der Statistik S handelt, soll diese Methode wahr zurückgeben.

gibSelektivität(Statistik S,KlassenId ID) gibt die gemessene oder berechnete Selektivität der Klasse ID in der Statistik S zurück.

setzGemesseneSel(S,ID,Sel) setzt die Klasse ID in der Statistik S auf die gemessene Selektivität Sel.

setzBerechneteSel(S,ID,Sel,N_{tief}, N_{hoch},#Klassen) setzt die Klasse ID in der Statistik S auf die berechnete Selektivität Sel und passt die Nachbarn N_{tief}, N_{hoch} und die Gesamtklassenanzahl $\#Klassen$ entsprechend an.

gibNachbarn(Statistik S,KlassenId ID) liefert die Nachbarn N_{tief}, N_{hoch} der Klasse ID in der Statistik S zurück.

gibGesamtKlassen(Statistik S,KlassenId ID) gibt die Gesamtklassenanzahl $\#Klassen$ der Klasse ID in der Statistik S zurück.

5.2 Algorithmen

In diesem Abschnitt werden die beiden Algorithmen $BerechneSel$, zur Berechnung der Selektivität für eine ausgewählt Menge von Klassen, und $AnwendungSel$, zum Anpassen der Statistik mit Hilfe des Wertes auf der Anfrageausführung, vorgestellt.

5.2.1 Algorithmus zur Berechnung der Selektivität

Mit Hilfe des Algorithmus $BerechneSel$ soll für eine gegebene Menge von Klassen einer Statistik S die Selektivität berechnet werden. Die Menge der Klassen, auf die unsere Anfrage abgebildet wurde, soll im folgenden immer mit $\mathcal{K}$ bezeichnet werden. Dabei gehen wir davon aus, dass $\mathcal{K}$ nur maximale Klassen enthält.

Algorithmus 4 BerechneSel

Eingabe: eine Statistik S und eine Menge von Klassen $\mathcal{K} = \{k_1, ..., k_n\}$

Ausgabe: die Gesamtselektivität für die Klassen $\mathcal{K} = \{k_1, ..., k_n\}$

```
Begin
 gesamtSel = 0;
 foreach k_i in K
  if( istVorhanden(S, k_i))
   gesamtSel = gesamtSel + gibSelektivität(S, k_i);
  else
   gesamtSel = gesamtSel+
    annahmeSelektivität(S, k_i);
  endIf
 endForeach
 return gesamtSel;
End
```

Im Algorithmus wird für jede Klasse aus $\mathcal{K}$ die Selektivität in der Statistik S ermittelt und aufsummiert. Wenn noch keine Selektivitätsinformation über die Klasse k_i existiert, wird mit Hilfe der Funktion $annahmeSelektivität$ die Selektivität anhand von statistischen Annahmen (z.B. Gleichverteilungsannahme) abgeschätzt. In diesem Fall ist es aber auch durchaus möglich, eine Funktion zu verwenden, die zuerst einmal versucht, die Selektivität aus der Menge von Teilklassen oder Oberklassen[1] von k_i zu bestimmen. Um eine korrekte Selektivität zu ermitteln ist es dabei notwendig, dass die Klassen in $\mathcal{K}$ sich nicht überschneiden. D.h. wenn die Klasse

[1]Oberklassen von k_i sind alle Klassen k in denen k_i Teilklasse ist.

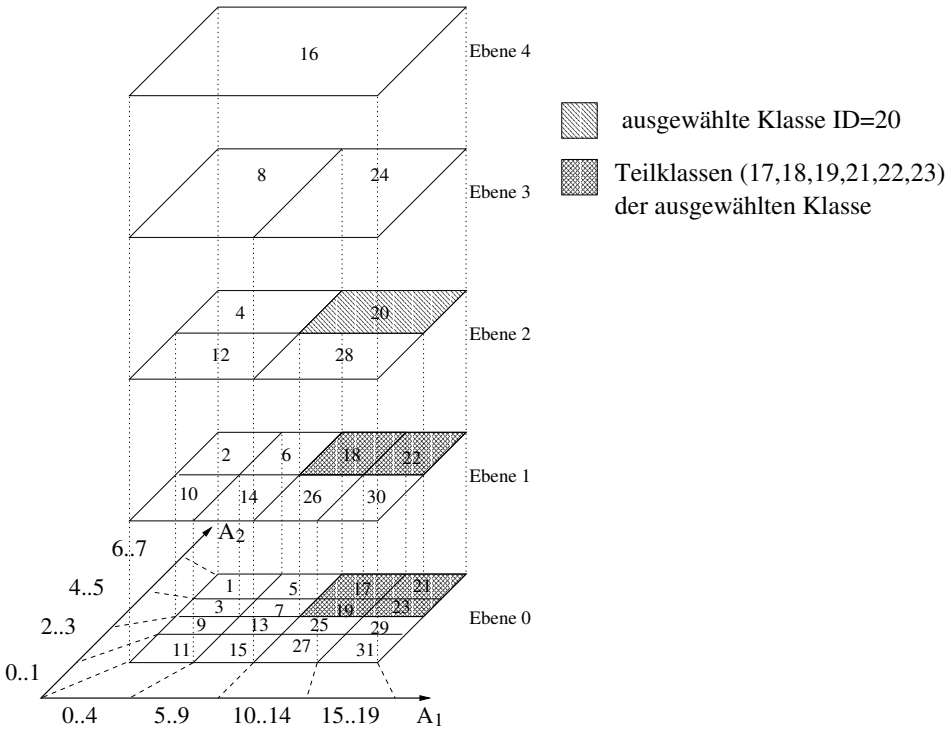

Abbildung 5.2: Beispiel für Teilklassen

k_i auf der Ebene e bereits in $\mathcal{K}$ ist, so darf keine Teilklasse von k_i der Ebenen $e' < e$ in $\mathcal{K}$ vorkommen. Auf Bild 5.2 ist zum Beispiel die Klasse mit der $ID = 20$ ausgewählt. Als Teilklassen ergeben sich daraus die Klassen mit den Bezeichnern $17, 18, 19, 21, 22\, und\, 23$.

5.2.2 Algorithmus zur Anpassung der Statistik

Der Algorithmus $AnwendungSel$ soll einen, von der Anfrageausführungseinheit ermittelten, Selektivitätswert auf eine Menge von Klassen einer Statistik anwenden bzw. verteilen. Bevor der Algorithmus beschrieben wird, werden noch einige Definitionen und Vorstellungen benötigt. Wie im vorher gehenden Abschnitt ist auch hier die Menge von Klassen aus der Anfrage mit $\mathcal{K}$ dargestellt. Der von der Anfrageausführungseinheit gelieferte Wert wird mit Sel bezeichnet. Im folgenden soll zuerst an einem kleinen Beispiel der schrittweise Aufbau der Statistik vorgestellt werden.

Danach soll dieser Vorgang verallgemeinert, die Erstellung und Behandlung von gemessenen Werten vorgestellt, und die 4 unterschiedlichen Fälle der Anpassung präsentiert werden.

Beispiel für den schrittweisen Aufbau der Statistik

Wie bereits in der Konzeptbeschreibung vorgestellt, wird am Anfang von einer komplett leeren Statistik (spärlich-adaptierbar) ausgegangen. Auf Bild 5.3 ist die Aufteilung der Klassen, ihre Nummerierung und die grafische Darstellung der Anfragen für die folgenden Beispiele abgebildet. Als erste Anfrage soll der selektierte Bereich ($5 \leq A_1 \leq 14\, und\, 2 \leq A_2 \leq 5$) der 1.Anfrage dienen. Daraus ergibt sich für $\mathcal{K}$ die Menge von $\{7, 13, 19, 25\}$ als Bezeichner der Klassen. Die Selektivität Sel soll S_1 sein. Da noch keine Informationen über die 4 Klassen vorhanden sind, wird die Selektivität, wie bereits in 4.2 vorgestellt, gleichmäßig auf die vier Klassen, unter Beachtung der Klassengröße, aufgeteilt. D.h. bei jeder Klasse wird die berechnete Selektivität auf $\frac{S_1}{4}$[2] gesetzt. Die vier Klassen werden mittels der Nachbarn entsprechend miteinander verknüpft. Solch eine verknüpfte Menge von Klassen soll im weiteren als Gruppe bezeichnet werden. Dabei kann eine Klasse in maximal einer Gruppe sein. Es können auch Gruppen mit nur einer einzelnen Klasse existieren. Die Bedingung an die Klassen einer Gruppe ist, dass alle ihre Selektivitäten sogenannte „berechnetet Selektivitäten" darstellen. Die Gruppe mit den Klassen $\{7, 13, 19, 25\}$ wird als G_1bezeichnet. Somit ist das Ende der Anpassung für die erste Anfrage erreicht. Eine folgende Anfrage, die gleich der 1.Anfrage ist, könnte somit mit der korrekten Selektivität[3] optimiert werden.

Bevor die 2.Anfrage bearbeitet wird, sollen noch einige weitere Notationen vorgestellt werden.

- $|G_i|$ - entspricht der Anzahl von Klassen in der Gruppe G_i.
- $|\mathcal{K}|$ - entspricht der Anzahl von Klassen in $\mathcal{K}$.

[2] alle Klassen haben $FE(k_i) = 1$

[3] unter der Annahme, dass sich die Selektivität nicht geändert hat

- $G_i[l]$ - ist die l-te Klasse aus G_i, mit $1 \leq l \leq |G_i|$.
- $Selek(k_i)$ - ist die gemessene oder berechnete Selektivität der Klasse k_i.
- $Selek(G_i)$ - ist gesamt Selektivität der in G_i enthaltenen Klassen, also $Selek(G_i) = \Sigma_{l=1}^{|G_i|} Selek(G_i[l])$.
- $Selek(\mathcal{K})$ - ist gesamt Selektivität aller Klassen $k_i \in \mathcal{K}$. Diese ist nicht zu verwechseln mit Sel, der Selektivität, die von der Anfrageausführungseinheit zurück geliefert wurde.
- $Fehler$ - als Fehler wollen wir die Differenz von Sel und $selek(\mathcal{K})$ bezeichnen.
- $FE(k_i)$ - stellt die Größe der Klasse k_i in Flächeneinheiten dar.
- $FE(G_i)$ - ist die Summe der Größen alle der Klasse k_i in G_i.
- $FE(\mathcal{K})$ - ist die Summe der Größen alle der Klasse $k_i \in \mathcal{K}$.

Als nächstes soll die Statistik mit Hilfe der 2. Anfrage angepasst werden. Für die 2.Anfrage soll dabei $Sel = S_2, \, mit \, S_2 < S_1$ und $\mathcal{K} = \{7, 13\}$ gelten. Bei der Anpassung wird festgestellt, dass für die Klassen $\{7, 13\}$ bereits berechnete Werte existieren und die beiden Klassen zur Gruppe G_1 gehören. Um die neue Selektivität auf die Klassen $\{7, 13\}$ anzupassen, muss der jeweilige berechnete Selektivitätswert $selek(k_i)$ aller Klassen $k_i \in \mathcal{K}$ auf $selek(k_i) = selek(k_i) + \frac{Fehler * FE(k_i)}{FE(\mathcal{K})}$ gesetzt werden. Im Beispiel muss also gerade $\frac{Fehler}{2}$ addiert werden. Der Fehler wird somit gleichmäßig, entsprechend der Größe auf die beteiligten Klassen, verteilt. Um auch die Selektivität S_1für die Klassen $\{7, 13, 19, 25\}$ korrekt darzustellen, müssen die Selektivitäten der Klassen $\{19, 25\}$ ebenfalls entsprechend verändert werden, $selek(k_i) = selek(k_i) - \frac{Fehler * FE(k_i)}{FE(G_1) - FE(\mathcal{K})}$. Neben den Anpassungen der Selektivitäten spalten wir auch die Gruppe $G_1 = \{7, 13, 19, 25\}$ in zwei Gruppen $G_1 = \{19, 25\}$ und $G_2 = \{7, 13\}$ auf. Unter der Annahme, dass sich die Verteilung seit der ersten Anfrage nicht verändert hat, wurde nun

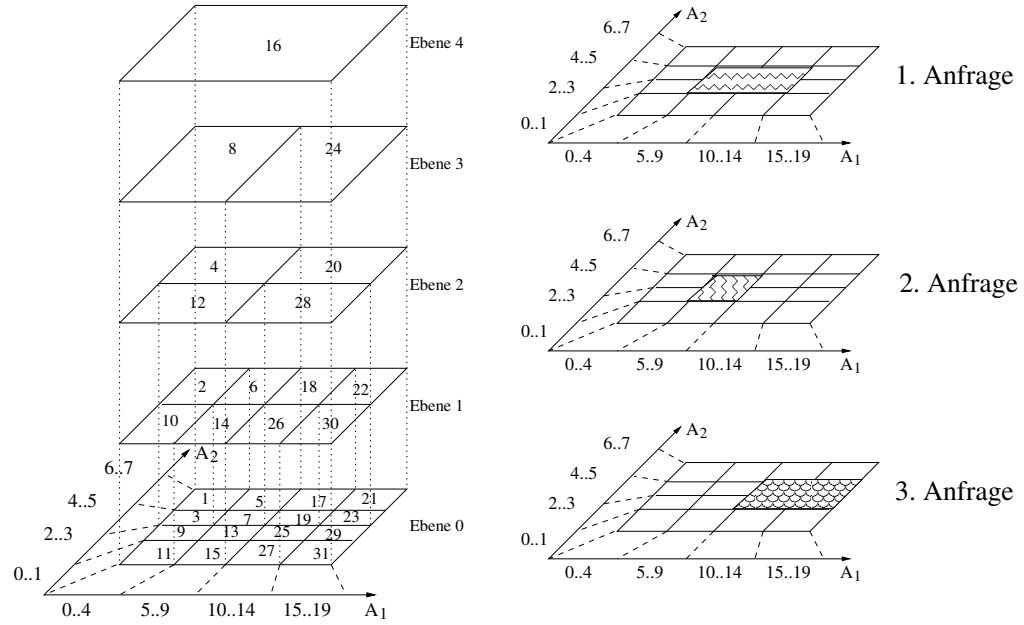

Abbildung 5.3: Anfragebeispiele

auch gleichzeitig die korrekte Selektivität für die Gruppe G_1 ermittelt oder gelernt. Man ist also in der Lage, nicht nur die 1. und 2. Anfrage mit korrekten Selektivitäten zu optimieren, sondern auch die Anfrage mit dem selektierten Bereich $10 \leq A_1 \leq 14 \, und \, 2 \leq A_2 \leq 5$.

Nun soll die Statistik mit der Selektivität S_3 aus der 3.Anfrage, $\mathcal{K} = \{19, 23, 25, 29\}$, angepasst werden. Man stellt fest, dass die Klassen 19 und 25 bereits durch die Gruppe G_1 dargestellt sind und $Selek(G_1)$ bereits korrekt ist. Daraus ergibt sich für die beiden Klassen 23 und 29 eine gesamt Selektivität von $S_3 - Selek(G_1)$. Somit wird jede der beiden Klassen mit der berechneten Selektivität $\frac{S_3 - Selek(G_1)}{2}$ initialisiert und es wird die Gruppe $G_3 = \{23, 29\}$ gebildet. Auch in diesem Beispiel konnten aus der neuen Selektivität S_3, die gesamt Selektivität für die Klassen 23 und 29 abgeleitet werden.

Klassen mit gemessener Selektivität

In Beispielen wurden immer nur berechnete Selektivitäten eingefügt. Nun stellt sich die Frage, wann werden gemessenen Selektivitäten eingefügt oder wann wird eine berechnete Klasse in eine gemessene umgewandelt. Nun, eine gemessene Selektivität soll die reale[4] Selektivität der Klasse darstellen. Sie muss also für diese eine Klasse gemessen worden sein. Daraus folgt, dass, wenn $\mathcal{K}$ genau eine Klasse enthält, diese Klasse den gemessenen Wert Sel erhält. Auf Bild 5.4 ist ein Beispiel für diesen Fall abgebildet. Dabei entspricht der Anfragebereich gleich $\mathcal{K} = \{4\}$ und somit würde die Klasse 4 auf die gemessene Selektivität Sel gesetzt werden.

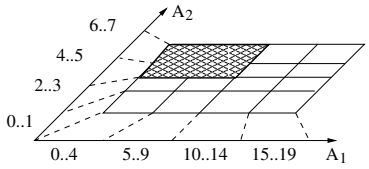

Abbildung 5.4: Gemessene Klasse, Beispiel

Ist die Klasse 4 dabei bereits in einer Gruppe von berechneten Werten vorhanden, muss die Gruppe um die Klasse 4 verringert werden. Weicht in diesem Fall die gemessenen Selektivität von der zuvor berechneten Selektivität der Klasse 4 ab, muss natürlich der Rest der Gruppe entsprechend angepasst werden.

Im Falle von statischen Verteilungen kann man eine einmal als gemessen abgespeicherte Selektivität für alle Zeiten als solche ansehen. Hat man es aber mit dynamischen Verteilungen zu tun, kann sie nicht für immer als solche angesehen werden, denn die Selektivität der Klasse könnte sich seit ihrer Messung geändert haben. Um diesem Umstand Rechnung zu tragen, sieht man eine gemessene Selektivität nur solange als real an, bis dieses widerlegt wurde, oder die Messung bereits zu lange zurück liegt. Dazu wird der Datenstruktur noch ein Zeitstempel hinzugefügt, der bei jedem Schreibzugriff auf die aktuellen Zeit gesetzt wird. Ist also ein ge-

[4] real zum Zeitpunkt der Einfügung

messener Wert nicht mehr real, wird er gelöscht, d.h. die Klasse wird aus der Statistik entfernt.

Es soll zuerst noch die Bedeutung von gemessenen Selektivitäten in diesem Zusammenhang erläutert werden, bevor die verschiedenen Fälle der Anpassung einer Statistik beschrieben werden. Dieses verdeutlicht folgendes Beispiel: Es sei die Klasse 4 mit einer Selektivität S_4gemessen und als solche in die Statistik eingefügt worden. Als nächste soll die Anfrage mit $\mathcal{K} = \{4, 9, 13\}$ und der Selektivität Sel in die Statistik eingefügt werden. Folgendes kann daraus geschlossen werden:

- wenn der Zeitstempel der gemessene Klasse 4 zu alt ist, muss vor der weiteren Anpassung die Klasse 4 gelöscht werden.
- wenn $Sel < S_4$, so steht das im Widerspruch zur gemessenen Klasse 4. Daher muss auch hier die Klasse vor jeder weiteren Anpassung gelöscht werden.
- falls keine der oberen beiden Bedingungen zutrifft, muss angenommen werden, dass die gemessene Selektivität der Klasse 4 auch ihre reale ist. Daraus ergibt sich dann die gesamt Selektivität $Sel - S_4$ für die beiden Klassen 9 und 13. D.h. man verringert in diesem Fall $\mathcal{K}$ um die Klasse 4 und die Selektivität Sel um S_4.

Daraus folgt für die allgemeine Anpassung, dass zuerst alle gemessenen Klassen, die in $\mathcal{K}$ enthalten sind, überprüft werden und die als gut befundenen (nicht gelöschten) gemessenen Klassen aus $\mathcal{K}$ gelöscht und deren Selektivitätswerte von Sel abgezogen werden.

Allgemeine Beschreibung der Anpassung

Die Beispiele aus dem schrittweisen Aufbau sollten zeigen wie der Ablauf bei der Anpassung der Statistik aussieht. Sie sollten aber auch verdeutlichen, dass es unterschiedliche Situationen bzgl. der bereits in der Statistik existierenden Klassen gibt. Bei der 1.Anfrage waren im Anpassungsbereich noch keine Klassen vorhanden, die 2.Anfrage fiel in eine bereits

existierende Gruppe von Klassen, und bei der 3.Anfrage war die Situation so, dass eine Gruppe in der Anfragebereich fiel, aber ihn nicht komplett ausfüllte, sondern auch neue Klassen erzeugt werden mussten.

Bei Betrachtung der möglichen Situationen für die Anpassungen und ihrer Behandlung, kristallisieren sich vier unterschiedliche Situationen der Handhabung heraus. Durch Kombination dieser 4 Situationen kann man dann alle in der Realität auftretenden Fälle bilden. Dabei sollen die gemessenen Klassen bereits aus $\mathcal{K}$ eliminiert und Sel entsprechend verringert sein. Diese 4 Fälle sind:

1. Fall (Bild 5.5 links, oben) - in diesem Fall sind alle Gruppen, bei denen mindestens eine Klasse in den Anfragebereich fällt, komplett im Anfragebereich enthalten. Außerdem existieren keine Klassen in $\mathcal{K}$ die noch nicht in der Statistik existieren,d.h. in keiner Gruppe sind. Mathematisch kann dieser Fall mit

$$für\, \forall G_i\, mit\, \mathcal{K} \cap G_i \quad \neq \emptyset\, muss\, gelten:$$

$$wenn\, k_m \in G_i,\ so\, k_m \in \mathcal{K}\, oder$$

$$\exists k_l \in \mathcal{K},\ so\, dass\, k_m Teilkasse\, von\, k_l$$

$$und\, \forall k_j \in \mathcal{K} \mid \exists G_i\, mit\, k_j \in G_i$$

charakterisiert werden. Im Fall einer statischen Verteilung gilt dann:

$$Selek(\mathcal{K}) = \sum Selek(G_i),\ für\, \forall G_i\, mit\, \mathcal{K} \cap G_i \neq \emptyset,$$

daher muss in diesem besonderen Fall keine Anpassung vorgenommen werden. Sonst wird für die Anpassung folgende Formel benutzt:

$$\forall k_i \in \mathcal{K},\ selek(k_i) = selek(k_i) + \frac{Fehler * FE(k_i)}{FE(\mathcal{K})}$$

Die Gruppen bleiben in ihrer Art weiter bestehen. Sie werden also nicht zusammengesetzt oder weiter aufgespalten.

2. Fall (Bild 5.5 rechts, oben) - dieser Fall ist wie der 1.Fall, nur dass hier Klassen in $\mathcal{K}$ existieren, die noch nicht in der Statistik sind. Die Charakterisierung hierfür ist:

$$für\, \forall G_i\, mit\, \mathcal{K} \cap G_i \quad \neq \emptyset\, muss\, gelten:$$

$$wenn\, k_m \in G_i,\ so\, k_m \in \mathcal{K}\, oder$$

$$\exists k_l \in \mathcal{K}, so\, dass\, k_m Teilkasse\, von\, k_l$$

$$und\, \exists k_j \in \mathcal{K} \mid \forall G_l\, gilt\, k_j \notin G_l$$

Bei der Anpassung muss dann bzgl. des Vergleiches $Sel \geq \sum selek(G_i)$ vorgegangen werden. Ist der Vergleich negativ, so ist ein Widerspruch aufgetreten und wir löschen alle Gruppen G_i. Danach wird eine neue Gruppe G mit $G = \mathcal{K}$ gebildet und der Wert Sel gleichmäßig auf die Klassen aus G verteilt. Mit

$$\forall k_i \in \mathcal{K},\ selek(k_i) = \frac{Sel * FE(k_i)}{FE(\mathcal{K})}.$$

Falls der Vergleich positiv war, nimmt man an, dass die Selektivität der bereits berechneten Flächen korrekt ist. Man bildet eine neue Gruppe mit den neuen Klassen aus $\mathcal{K}$. Der Fehler wird gleichmäßig auf die m-neuen Klassen, mit $selek(k_i) = \frac{Fehler * FE(k_i)}{\sum FE(k_m)}, 1 < i < m$, verteilt.

3. Fall (Bild 5.5 links, unten) - hier sind, wie im ersten Fall, keine Klassen in $\mathcal{K}$, die noch nicht in der Statistik sind, also $\forall k_j \in \mathcal{K} \mid \exists G_i\, mit\, k_j \in G_i$. Aber alle Gruppen, die in $\mathcal{K}$ auftreten, haben mindestens eine Klasse, die außerhalb von $\mathcal{K}$ liegt:

$$für\, \forall G_i\, mit\, \mathcal{K} \cap G_i \quad \neq \emptyset\, muss\, gelten:$$

$$\exists k_m \in G_i,\ so\, dass\, k_m \notin \mathcal{K}\, und$$

$$\forall k_l \in \mathcal{K},\ gilt\, k_m\, nicht\, Teilklasse\, von\, k_l$$

In diesem Fall muss bzgl. des Vergleichs $Sel \leq \sum selek(G_i)$ unterschieden werden. Auch in diesem Fall ist bei einem negativen Ergebnis ein Widerspruch festgestellt worden. Man geht dann wie im 2.Fall bei einem negativen Ergebnis vor. Im positiven Fall muss der Fehler entsprechend aufgeteilt werden. Die Gruppen werden weiter aufgespalten in Gruppen aus Klassen, die in $\mathcal{K}$ liegen und die, die es nicht tun. Bei der Aufteilung des Fehlers kann die Formel aus dem ersten Fall angewendet werden.

$$\forall k_i \in \mathcal{K},\ selek(k_i) = selek(k_i) + \frac{Fehler * FE(k_i)}{FE(\mathcal{K})}$$

Andererseits müssen auch die restlichen Klassen, die außerhalb von $\mathcal{K}$ lagen, angepasst werden, mit

$$selek(k_i) = selek(k_i) - \frac{Fehler * FE(k_i)}{\sum FE(G_i) - FE(\mathcal{K})}$$

4. Fall (Bild 5.5 rechts, unten) - der letzte Fall, ist wie der 3-te Fall gelagert, nur dass die Einschränkung bezüglich der noch nicht vorhandenen Klassen in der Statistik nicht gilt. In $\mathcal{K}$ ist also mindestens eine Klasse enthalten, die noch nicht in der Statistik enthalten ist. Bei der Anpassung wird eine Kombination aus Fall 2 und Fall 3 angewandt.

Wie bereits beschrieben, können alle möglichen auftretenden Fälle als Mischfälle der oberen 4 dargestellt werden. Dabei können speziell die Fälle 1 und 3 und die Fälle 2 und 4 zusammen auftreten. In diese Fällen wird nach den entsprechenden Mustern der enthaltenen Fälle vorgegangen. Dabei werden immer die Fälle 1 und 2 zuerst bearbeitet und dann die Fälle 3 und 4.

Als letztes soll noch ein besonderer Fall besprochen werden. Der Fall, in dem die Selektivität, die von der Anfrageausführungseinheit zurückgeliefert wurde, gleich Null ist. In diesem Fall können wir für alle Klassen in $\mathcal{K}$, eine gemessene Selektivität von Null einsetzen und dieses auch für alle Teilklassen schlussfolgern. Hierbei ist nur zu beachten, dass für bereits be-

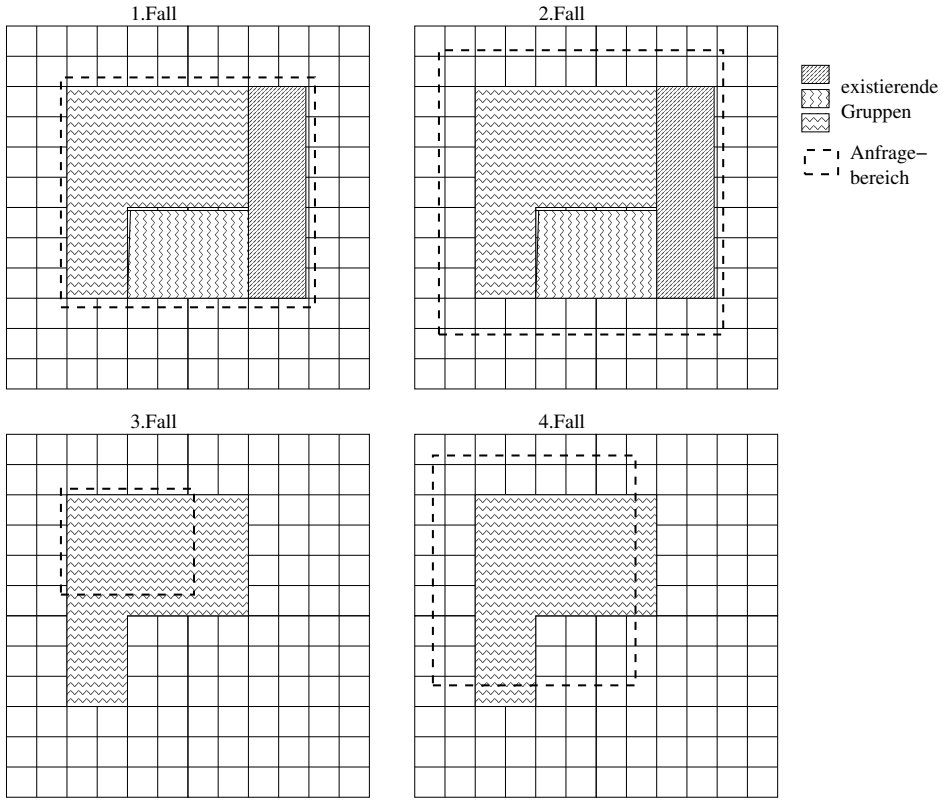

Abbildung 5.5: 4 Fälle der Anpassung

rechnete Klassen in $\mathcal{K}$ auch die Gruppen entsprechend angepasst werden müssen.

Kapitel 6

Die Wahl der Partitionsfunktion

Im ersten Abschnitt von diesem Kapitel soll eine kurze Einführung zur Z-Ordnung und zu balancierten Bäume gegeben werden. Dabei werden die Partitionsfunktion P und die Abbildungsfunktion Abb vorgestellt. Im zweiten Abschnitt wird dann die Umsetzung des Konzepts mit Hilfe von balancierten Bäumen und der Z-Ordnung vorgestellt und begründet, sowie die im Abschnitt 4.2 aufgestellten Anforderungen an das Konzept mit den Eigenschaften der Umsetzung kritisch verglichen. Im Kapitel 7 wird dann die Implementierung, die im Rahmen dieser Diplomarbeit entstand, vorgestellt.

6.1 Einführung in Z-Ordnung und B-Bäume

6.1.1 Balancierte Bäume

Balancierte Bäume (B-Bäume) ([Fre00, Fre02]) sind eine spezielle Art der Mehrwegebäume. Ihre spezielle Eigenschaft ist ihre Balanciertheit. D.h. der Pfad von der Wurzel des Baumes bis zu jedem Blatt ist gleich lang. B-Bäume werden in DBMSen hauptsächlich als Indexstrukturen genutzt. Ihre Bedeutung liegt darin, dass B-Bäume bereits eine Sekundärspeicherstruktur darstellen. Es ist also möglich, einen B-Baum, oder Teile von ihm, auf den Sekundärspeicher auszulagern. Dabei wird je ein Knoten des B-Baumes auf je eine Seite des Sekundärspeichers geschrieben. Somit müs-

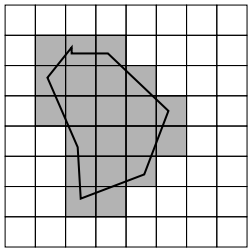

Abbildung 6.1: Approximation durch Gitterelemente

sen nur die Knoten des Baumes in den Hauptspeicher geladen werden, auf die auch wirklich zugegriffen wird. Bei der Suche ist im Durchschnitt, und auch im schlechtesten Fall, der Zugriff auf $log_{k+1}((n+1)/2)$ Knoten erforderlich. k stellt dabei den Mindestfüllgrad der internen Knoten und Blätter dar. n dagegen ist die Gesamtanzahl der Einträge im Baum. B-Bäume präsentieren daher eine effiziente Sekundärspeicherstruktur.

6.1.2 Z-Ordnung

In vielen Anwendungen ist es notwendig, den Werteraum[1] zu diskretisieren. So auch bei unserer Anwendung des Lernens von Statistiken. Dabei wird über den Werteraum ein regelmäßiges Gitter gelegt. Anfragen werden dann durch geeignete Mengen von Gitterelementen approximiert (Abb. 6.1). Ein entscheidender Faktor ist die Wahl der Gittergröße oder Gittergranularität. Sie beeinflusst einerseits die Genauigkeit der Approximation und zum anderen die Größe der Menge der zur Approximation gehörenden Gitterelemente. [Fla97]

Um diese Menge auf einer effizienten Art und Weise zu speichern und zu bearbeiten, werden Funktionen genutzt, die eine Aufzählung der einzelnen Gitterelemente ermöglichen. Es wird somit eine Ordnung auf die Menge aller Elemente definiert, die durch die Aufzählung implizit gegeben ist. Somit wird der Werteraum diskretisiert. Die Aufzählungsfunktion dagegen ermöglicht eine Abbildung aus 2- bzw. n-Dimensionen auf eine

[1] Definition - Seite 43

Dimension. Man erhält somit eine totale Ordnung auf der Menge von Gitterelementen.

Bei dieser Aufzählungsfunktion kommt die Z-Ordnung (engl. Z-Ordering) ins Spiel. Oft ist es notwendig, mit Hilfe dieser Funktion Nachbarschaftsbeziehungen möglichst gut darzustellen. Diese Art von Funktionen werden raumfüllende Kurven genannt. Zwei prominente Vertreter sind die Hilbertkurve und die Peanokurve, die auch Z-Ordering genannt wird (Abb. 6.2). Die Aufzählungsfunktion kodiert somit ein Gitterelement auf eine natürliche Zahl.

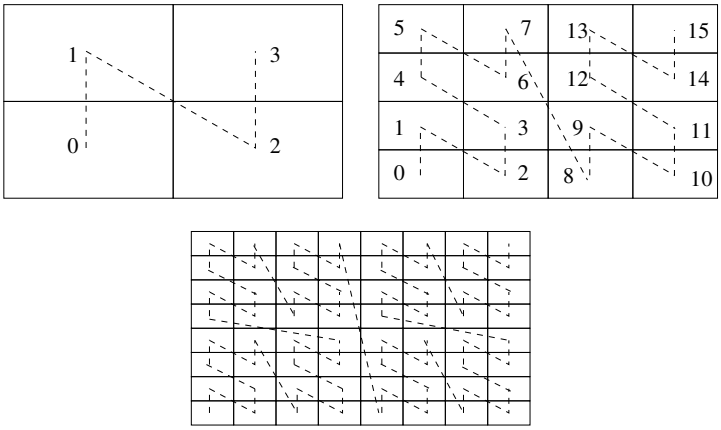

Abbildung 6.2: Z-Ordnung: Aufzählung der Gitterelemente bei unterschiedlicher Gitterauflösung (Quelle [Fla97])

Approximation eines Objektes/Anfrage im Werteraum

Bei der Approximation der Objekte, oder in unserem Fall der Anfrage im Werteraum, werden rekursiv die z-Werte bestimmt. Dabei wird ein z-Wert solange geteilt, bis entweder die maximale Granularität erreicht ist, oder sich keine Kante des Objektes mehr im z-Wert befindet. Der entsprechende Algorithmus wird in [Fla97, Ore86, Ore89] angegeben. Durch diese Methode der z-Werte Bestimmung entstehen Werte mit variablen Bit-Stellen. Variable Bit-Stellen sind im allgemeinen schwer in DBMSen zu handhaben.

Um dieses Problem zu umgehen, existiert ein Verfahren zur Approximation, bei dem z-Werte mit fester Anzahl von Bit-Stellen verwendet werden. Es werden Ebenen verschiedener Granularität gebildet, um daraufhin die z-Werte der einzelnen Gitterelemente zu berechnen. Diese Berechnung soll dann ebenfalls die geometrischen Eigenschaften der einzelnen Elemente darstellen. Auf Abbildung 4.4 auf Seite 45 sind die Ebenen, Gitterelemente (Klassen) und ihre z-Werte abgebildet.

Partitionsfunktion

Die Partitionsfunktion P soll den Werteraum auf n-Ebenen aufteilen und die entsprechenden z-Werte bestimmen. Dabei haben alle z-Werte die konstante Bitlänge von n. Zur Berechnung wird dabei rekursiv von Ebene $n-1$ bis auf Ebene 0 heruntergegangen. Zur Bestimmung der z-Werte setzten wir als erstes den z-Wert der Klasse in der Ebene $n-1$ auf 2^{n-1}. Danach halbiert man die zuvor berechnete Klasse (Vaterklasse) abwechselnd auf der Dimension der beiden Attribute A_1, A_2. Die so erzeugte linke und rechte Klasse auf der Ebene e erhalten den z-Wert $zWert(Vater) \pm 2^e$. Diese Rekursion wird so lange vorgesetzt bis $e = 0$ ist.

Abbildungsfunktion

Die Abbildungsfunktion

$$Abb = Abb(min_Q(A_1), min_Q(A_2), max_Q(A_1), max_Q(A_2))$$

soll einen Bereich in Werteraum der Attribute A_1 und A_2auf die entsprechende Menge von z-Werten abbilden. Die Menge von z-Werten soll die minimale Menge darstellen. Bei der Berechnung dieser Menge kann man entweder von oben nach unten gehen (TOP-DOWN), oder man geht von Ebene 0 nach oben (BOTTOM-UP). Im folgenden soll die Methode BOTTOM-UP vorgestellt werden. Dabei berechnet man zuerst alle z-Werte der Ebene 0 für die Klassen, in die der Anfragebereich fällt (Alg. 5).

Im Algorithmus 5 ist mit der Funktion P, die Partitionsfunktion ge-

Algorithmus 5 Berechnung der z-Werte der Ebene 0

Eingabe: $min_{A_1}, min_{A_2}, max_{A_1}, max_{A_2}$

Ausgabe: Menge $\mathcal{K}$ von z-Werten der Ebene 0

```
for(a1 = min_A1; a1 ≤ max_A2; a1 = a1 + Klassenbreite_A1)
 for(a2 = min_A2; a2 ≤ max_A2; a2 = a2 + Klassenbreite_A2)
   zWert = P(a1, a2, e = 0);
   K = K ∪ {zWert};
 endfor
endfor
```

meint. Sie soll den z-Wert für die Position (a_1, a_2) auf der Ebene 0 bestimmen. Auf die Menge $\mathcal{K}$, die der Algorithmus zurückgeliefert hat, wendet man nun den Algorithmus $Komprimieren$ an. Dieser Algorithmus versucht je zwei Werte aus $\mathcal{K}$ zu einem neuen z-Wert einer höheren Ebene zu verschmelzen. Für die Verschmelzung zweier z-Werte (z_1, z_2) zu $z_{neu} = (z_1 + z_2)/2$, müssen folgende Bedingungen gelten:

1. z_1 und z_2 liegen auf der gleichen Ebene (e)
2. $|z_1 - z_2| = ((z_1 + z_2)/2) mod (2^{e+1})$

Die Verschmelzung muss solange anhalten, bis sich keine neuen Werte mehr ergeben. Mit den beiden Algorithmen und der Partitionsfunktion P ist es also möglich, die minimale Menge von z-Werten zu bestimmen.

6.2 Umsetzung mit Z-Ordnung und B-Bäumen

Wie im 1. Abschnitt vorgestellt, ermöglicht die Verwendung von B-Bäumen eine effiziente Zugriffsstruktur, die auch einfach auf den Sekundärspeicher auszulagern ist. Um B-Bäume benutzen zu können, benötigt man eine Abbildung unseres Werteraumes auf eine Dimension. Dabei hilft die Z-Ordnung, denn sie bildet den Werteraum in eine Dimension ab. Allein diese Abbildung könnte auch jede andere Funktion liefern. Der Vorteil der

Z-Ordnung liegt in der Beibehaltung der Lokalität bzw. der räumlichen Eigenschaften des Werteraumes. D.h., Klassen die räumlich nahe beieinander liegen, haben ähnliche z-Werte. Diese Lokalität wird speziell durch den B-Baum ausgenutzt. Ähnliche Klassen liegen dann entweder im gleichen oder in nahe beieinander liegenden Knoten. Das reduziert den Sekundärspeicherzugriff.

Bei der Umsetzung meines Konzeptes wird für jede Statistik ein B-Baum benutzt, indem die einzelnen Klassen mit ihrem Inhalt in den Knoten gespeichert werden. Die Aufteilung und Benennung der Klassen wird durch die Z-Ordnung auf n-Ebenen mit konstanten Bit-Stellen erstellt.

Als letztes sollen nun die Eigenschaften der Umsetzung mit den vier Bedingungen aus 4.2 verglichen werden. Die Anforderungen waren:

1. Strukturen, die nicht nur eine Hauptspeicherstruktur darstellen, sondern auch auf den Sekundärspeicher ausgelagert werden können.
2. schneller und effizienter Zugriff auf die Daten.
3. pro Relation so wenig Statistiken wie möglich. Diese Statistiken sollen über alle Teilmengen ihrer Attribute Aussagen zulassen.
4. die Nutzung von Aussagen oder Ergebnissen der Anfrageausführung durch Rückkopplung.

Durch die Nutzung der B-Baumstruktur wird der erste und zweite Punkt erfüllt. Im dritten Punkt kann nur der zweite Teil erfüllt werden, da die Einschränkung auf 2-dimensionale Statistiken getroffen wurde. Durch den Ansatz des Konzeptes, des Lernens von Statistiken durch Rückkopplung, wird der vierte Punkt beantwortet.

Kapitel 7

Beispielimplementierung

Im Kapitel 7 soll kurz, die im Rahmen dieser Diplomarbeit entstandene Software, vorgestellt werden. Dabei wird der Hauptteil dem Abschnitt 7.2 gehören, indem vorgenommene Messungen dargestellt werden. Im letzten Abschnitt werden Probleme und Beobachtungen, die bei der Implementierung gemacht wurden, vorgestellt.

7.1 Vorstellung der Programme

Bei der Implementierung des Konzepts wurde die Programmiersprache *C++* verwendet. Die graphischen Oberflächen wurden mit Hilfe der Qt-Bibliotheken, der Firma Trolltech, erstellt. Im Rahmen der Diplomarbeit entstanden dabei drei Programme. Das erste Programm QTDIST (Abb. 7.2), dient dazu mit Hilfe einer graphischen Oberfläche, eine 2-dimensionale Datenverteilung zu erstellen und in einer Datei abzuspeichern. Diese erstellten Verteilungen dienen dann in den beiden anderen Programmen als Berechnungsgrundlage. Sie soll somit eine reale Verteilung darstellen. Das Programm QTBTREE ist dazu gedacht, den schrittweisen Aufbau der Statistik zu visualisieren. Dazu existieren folgende 3 Sichten auf die aktuelle Statistik:

1. die Darstellung des aktuellen balancierten Baumes, wobei die Information jedes einzelnen Knoten angezeigt werden kann (Abb. 7.3).

2. die Sicht auf die n-Ebenen der Z-Ordnung und die sich ergebenen Klassen (Abb. 7.4).

3. eine Abbildung der untersten Ebene, also die Klassen mit der Flächengröße 1 (Abb. 7.5).

Mit Hilfe der zweiten und dritten Sicht können Anfragen simuliert werden und die Statistik entsprechend verändert werden. Das letzte Programm QTSTAT ist dazu erstellt worden, um einen Vergleich von verschiedenen Methoden der Selektivitätsabschätzung zu erhalten. Bei QTSTAT dient eine statische Verteilung als Grundlage für die Abschätzungen. Es kann eine Menge von Anfragen definiert werden. Aus dieser Menge wird dann per Zufall eine Anfrage ausgewählt und die Selektivitätsabschätzung für diese Anfrage erstellt. Das Ergebnis der jeweiligen Abschätzung wird mit dem realen Wert aus der Verteilung verglichen, der quadratische Fehler berechnet, und in einem Diagramm dargestellt. Zur Abschätzung der Selektivität werden drei verschiedene Methoden genutzt. Diese sind:

- Gleichverteilungsannahme, Klassen gleicher Größe haben immer die gleiche Selektivität.
- 1-dimensionale Histogramme und die Annahme der Unabhängigkeit der Attribute. Dabei spiegeln die Histogramme immer die aktuelle 1-dimensionale Verteilung wider.
- und das von uns entwickelte Konzept zum lernen von Statistiken.

Auf Bild 7.1 auf der nächsten Seite ist eine Ausgabe von QTSTAT abgebildet.

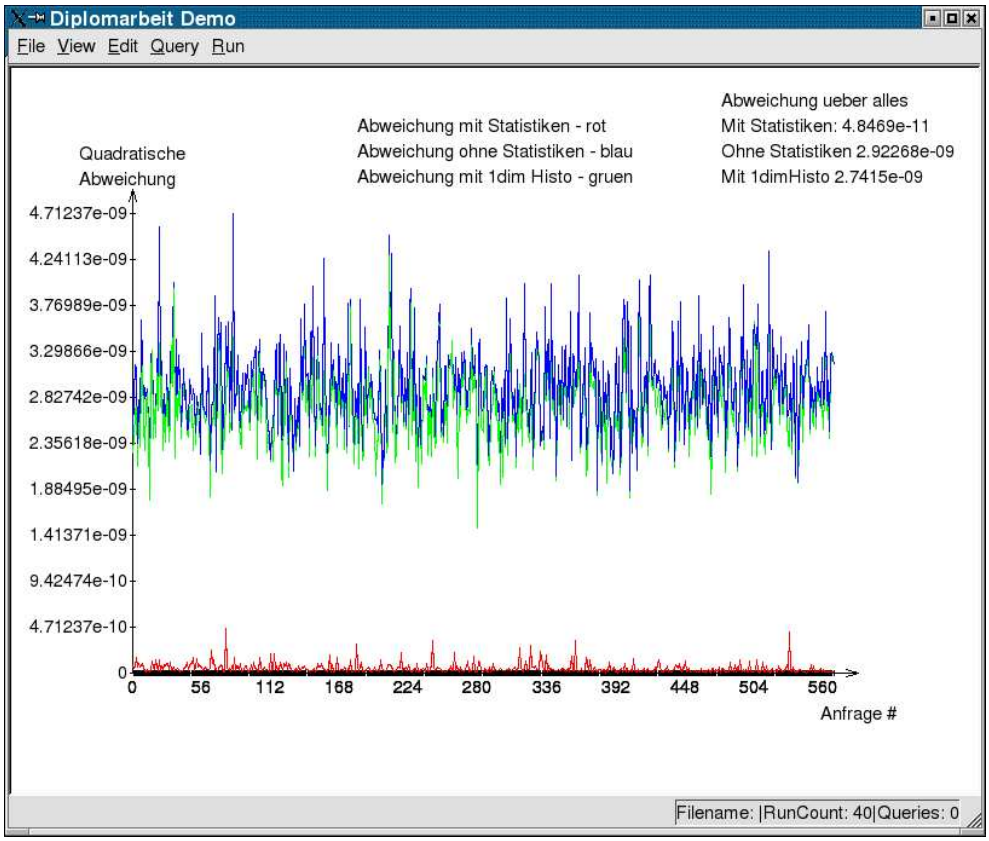

Abbildung 7.1: QTSTAT

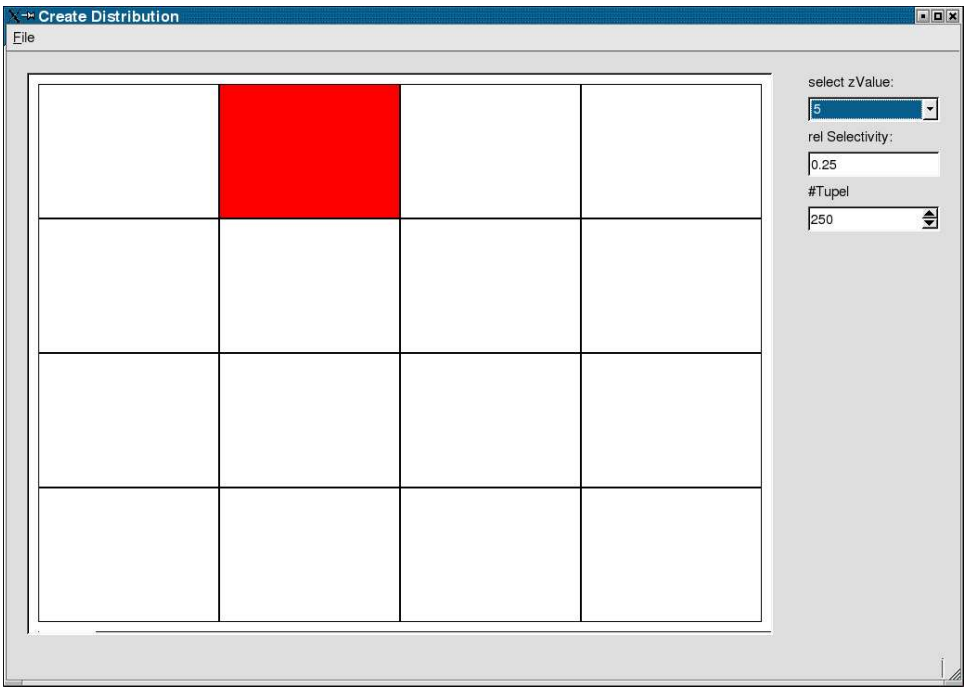

Abbildung 7.2: Programmoberfläche - QTDIST

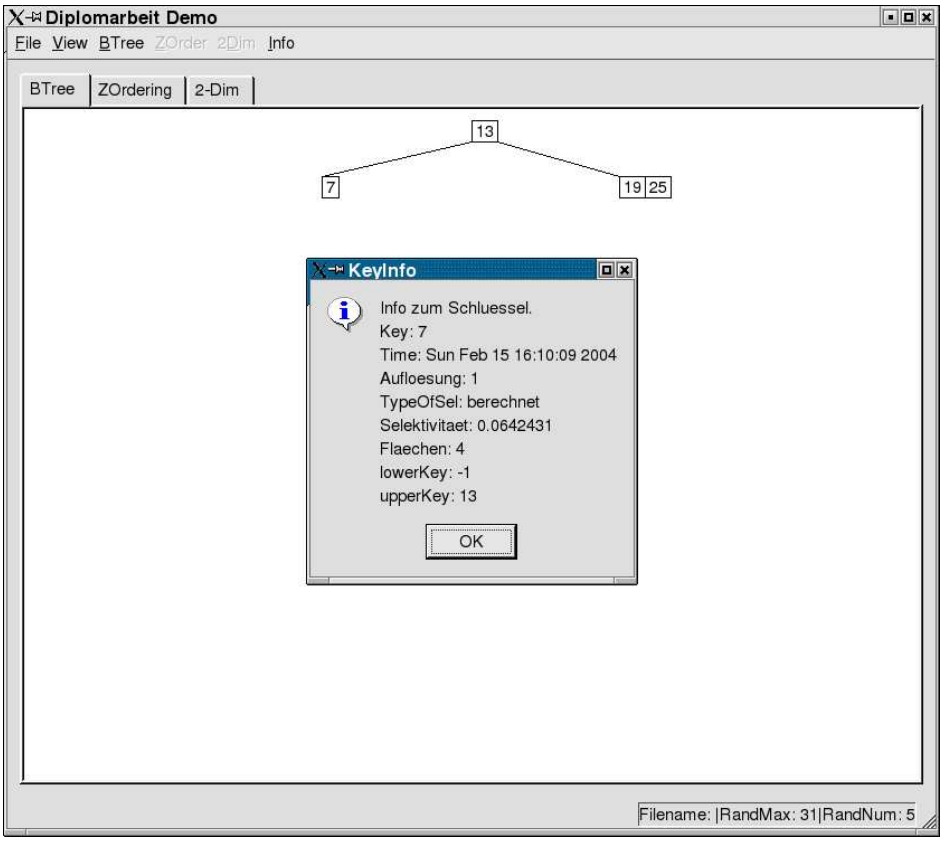

Abbildung 7.3: Programmoberfläche - QTBTREE

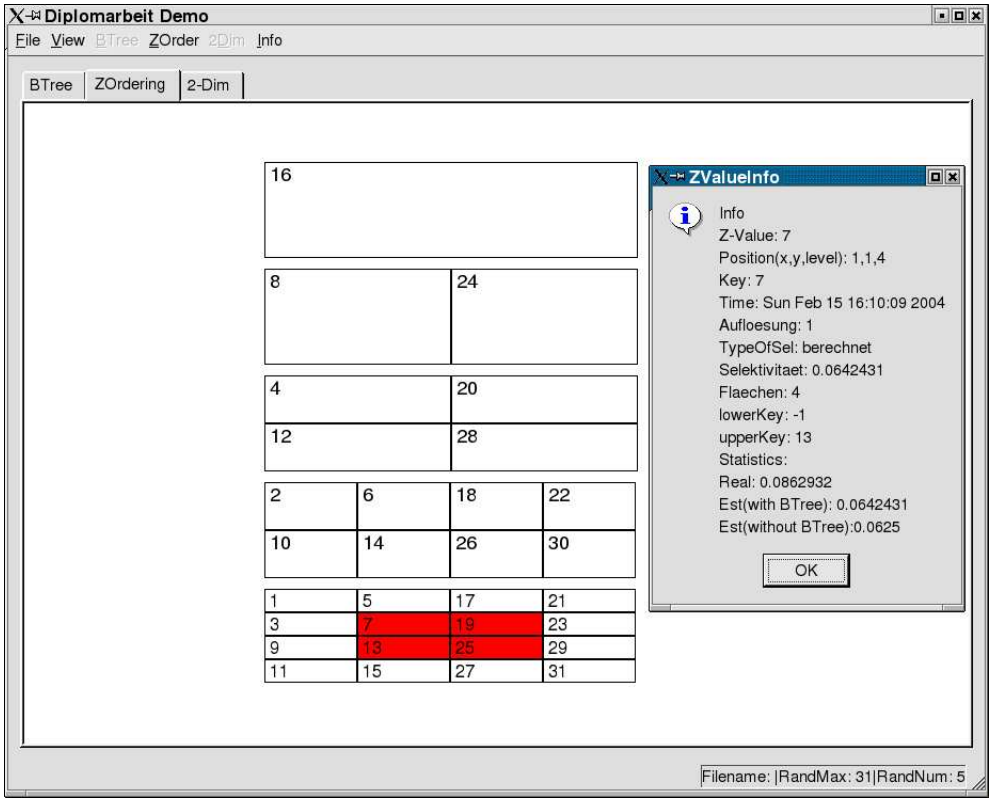

Abbildung 7.4: Programmoberfläche - QTBTREE

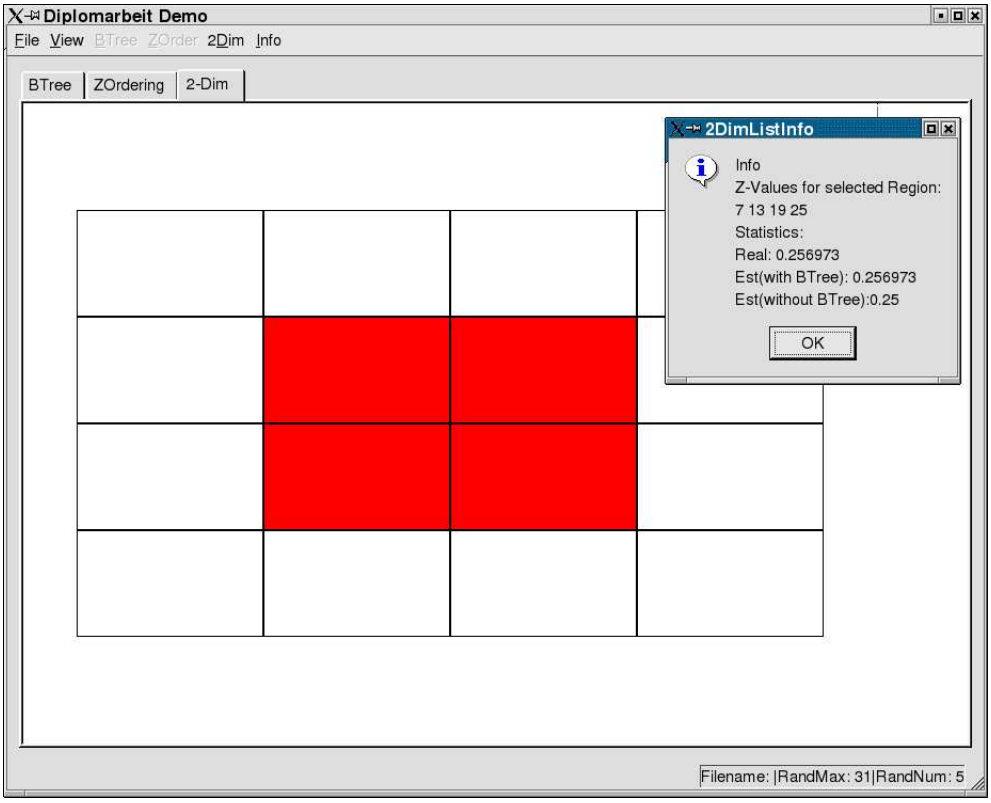

Abbildung 7.5: Programmoberfläche - QTBTREE

7.2 Messungen

In diesem Abschnitt werden Messungen vorgestellt, die mit Hilfe des Programmes QTSTAT gewonnen wurden. Dazu wurde das Programm insofern abgeändert, dass die Menge von Anfragen per Zufall erstellt wird. Die Messungen wurden unter folgenden Bedingungen gemacht:

- Computer: ein Athlon 1400++ mit 512MB Hauptspeicher
- Betriebssystem: SuSE-Linux, Kernel 2.4.21-99-athlon
- Datenverteilung entspricht der PartSupplier Relation eines TPC-H - 1GB Benchmarks, Version 2.1.8.1 (Quelle [TPC04]). Die Relation enthält dabei 200000 Tupel mit 200000 verschiedenen Teilen(Parts) und 10000 Lieferanten(Suppliers). Für die Partitionierung wurde ein Z-Ordering mit 21 Ebenen gewählt, das entspricht 1024x1024 Klassen auf Ebene 0.
- die Menge von Anfragen war 100000 Stück groß, dabei war eine Anfrage durchschnittlich 80 Flächeneinheiten groß und hatte eine reale Selektivität von durchschnittlich $15\frac{Tupel}{Anfrage}$.

In Tabelle 7.1 sind die Ergebnisse der Messungen dargestellt. Die Spalte Anfragen gibt dabei an, wieviele Anfragen zur Berechnung des durchschnittlichen Fehlers an das System gestellt wurden. Die Spalte Lernphase gibt an, wieviele Anfragen bereits vor der eigentlichen Messung an das System gestellt wurden. Die Anfragen aus der Lernphase wurden nicht aus der Menge der 100000 Anfragen entnommen, sondern wurden per Zufall erzeugt. Die letzten drei Spalten geben für die jeweilige Methode der Selektivitätsabschätzung den durchschnittlichen, quadratischen Fehler pro Anfrage an. Zur besseren Verständlichkeit wurden die Fehlerquadrate noch auf die Tupelanzahl umgerechnet. Die durchschnittliche Laufzeit pro Anfrage betrug bei den Messungen $0.6ms$.

Tabelle 7.2 zeigt die Ergebnisse der Messungen, bei denen die Anfragen durchschnittlich 280 Flächeneinheiten groß waren und eine reale Se-

Anfragen	Lernphase	Statistiken lernen	Gleichverteilungsannahme	1-dim. Histogramme
100000	0	$1.68E^{-9} \approx 8$	$2.94E^{-9} \approx 11$	$2.76E^{-9} \approx 11$
200000	0	$1.14E^{-9} \approx 6$	$2.94E^{-9} \approx 11$	$2.76E^{-9} \approx 11$
500000	0	$5.38E^{-10} \approx 4$	$2.94E^{-9} \approx 11$	$2.76E^{-9} \approx 11$
100000	50000	$1.43E^{-9} \approx 7$	$2.94E^{-9} \approx 11$	$2.76E^{-9} \approx 11$
200000	50000	$9.8E^{-10} \approx 6$	$2.94E^{-9} \approx 11$	$2.76E^{-9} \approx 11$
500000	50000	$4.6E^{-10} \approx 4$	$2.94E^{-9} \approx 11$	$2.76E^{-9} \approx 11$
100000	100000	$1.25E^{-9} \approx 6$	$2.94E^{-9} \approx 11$	$2.76E^{-9} \approx 11$
200000	100000	$8.62E^{-10} \approx 5$	$2.94E^{-9} \approx 11$	$2.76E^{-9} \approx 11$
500000	100000	$4.0E^{-10} \approx 3$	$2.94E^{-9} \approx 11$	$2.76E^{-9} \approx 11$

Tabelle 7.1: Messungen

lektivität von 52 hatten. Die Verteilung und Anfragenmenge entsprachen denen der Messungen zuvor.

Als letzte Messung wurde gegeüber der Ersten nur die Verteilung geändert. Die Veränderungen in der Verteilung wurden so gewählt, dass aus der ursprünglichen Verteilung 100 Klassen mit einer extrem hohem Selektivität hervorgehen. Die Tabelle 7.3 zeigt die entsprechenden Ergebnisse.

Anfragen	Lernphase	Statistiken lernen	Gleichverteilungsannahme	1-dim. Histogramme
100000	0	$5.62E^{-9} \approx 15$	$1.01E^{-8} \approx 20$	$8.50E^{-9} \approx 18$
200000	0	$3.88E^{-9} \approx 12$	$1.01E^{-9} \approx 20$	$8.50E^{-9} \approx 18$
500000	0	$1.90E^{-9} \approx 9$	$1.01E^{-8} \approx 20$	$8.50E^{-9} \approx 18$
100000	50000	$6.14E^{-9} \approx 16$	$1.01E^{-8} \approx 20$	$8.50E^{-9} \approx 18$
200000	50000	$4.16E^{-9} \approx 13$	$1.01E^{-8} \approx 20$	$8.50E^{-9} \approx 18$
500000	50000	$1.90E^{-9} \approx 9$	$1.01E^{-8} \approx 20$	$8.50E^{-9} \approx 18$
100000	100000	$5.87E^{-9} \approx 15$	$1.01E^{-8} \approx 20$	$8.50E^{-9} \approx 18$
200000	100000	$4.03E^{-9} \approx 13$	$1.01E^{-8} \approx 20$	$8.50E^{-9} \approx 18$
500000	100000	$1.84E^{-9} \approx 9$	$1.01E^{-8} \approx 20$	$8.50E^{-9} \approx 18$

Tabelle 7.2: Messungen

Anfragen	Lernphase	Statistiken lernen	Gleichverteilungsannahme	1-dim. Histogramme
100000	0	$1.70E^{-9} \approx 8$	$3.00E^{-9} \approx 12$	$2.81E^{-9} \approx 11$
200000	0	$1.15E^{-9} \approx 7$	$3.00E^{-9} \approx 12$	$2.81E^{-9} \approx 11$
500000	0	$5.44E^{-10} \approx 5$	$3.00E^{-9} \approx 12$	$2.81E^{-9} \approx 11$
100000	50000	$1.44E^{-9} \approx 8$	$3.00E^{-9} \approx 12$	$2.81E^{-9} \approx 11$
200000	50000	$9.88E^{-10} \approx 6$	$3.00E^{-9} \approx 12$	$2.81E^{-9} \approx 11$
500000	50000	$4.6E^{-10} \approx 4$	$3.00E^{-9} \approx 12$	$2.81E^{-9} \approx 11$
100000	100000	$1.27E^{-9} \approx 7$	$3.00E^{-9} \approx 12$	$2.81E^{-9} \approx 11$
200000	100000	$8.68E^{-10} \approx 6$	$3.00E^{-9} \approx 12$	$2.81E^{-9} \approx 11$
500000	100000	$4.14E^{-10} \approx 4$	$3.00E^{-9} \approx 12$	$2.81E^{-9} \approx 11$

Tabelle 7.3: Messungen

Kapitel 8

Zusammenfassung und Ausblicke

Mit dieser Arbeit sollten die Möglichkeiten des maschinellen Lernens zur Leistungssteigerung in DBMSen überprüft werden. Im ersten Teil wurde aufgezeigt, was maschinelles Lernen auf dem Gebiet der KI bedeutet. Danach wurden die Möglichkeiten des Lernens in DBMSen untersucht. Dabei habe ich bereits existierende Projekte vorgestellt und aufgezeigt, wie Methoden der KI zur Anwendung kommen. Man konnte dabei beobachten, dass diese Form des Lernens auf dem Gebiet der DBSe noch nicht ausreichend entwickelt ist. Ich stellte auch zwei Projekte vor, die zur Verbesserung der Anfrageoptimierung bzw. der Selektivitätsberechnungen dienten. Aus diesen zwei Projekten entwickelte sich die Idee vom Lernen von Statistiken durch Rückkopplung. Dies wurde weiter zu einem Konzept ausgebaut, welches es ermöglichte, mehr dimensionale Statistiken zu erlernen. Trotz der Einschränkungen auf 2-Dimensionen konnte ich mit Hilfe der Implementierung meines Konzeptes eine Verbesserung der Selektivitätsabschätzung aufzeigen. Insbesondere stellte ich dieses bei Anfragen, die mehrmals liefen, fest.

Die Annahme oder Hoffnung, dass sich bereits nach den ersten Anfragen ein Restfehler von Null ergibt, konnte nicht erfüllt werden. Dieses muss hauptsächlich der nicht vollständigen Implementation zugeschrieben werden. Es ist anzunehmen, dass bei einer Implementation des gesamten Konzeptes, mit allen möglichen Feinheiten, sich ein wesentlich

schnellerer Lernerfolg einstellt.

Die Möglichkeiten einer anderen Umsetzung des Konzeptes, z.B. durch UB-Bäume, sind gegeben und sollten in der Zukunft überprüft werden. Auch die Erweiterung auf höher dimensionale Statistiken ist sinnvoll und vom Konzept her machbar. Im Sinne des Lernens in DBMSen steht man gerade am Anfang, und es ergeben sich auf den verschiedensten Gebieten die Möglichkeiten vom maschinellen Lernen Gebrauch zu machen.

Literaturverzeichnis

[AC99] Ashraf Aboulnaga and Surajit Chaudhuri. Self-tuning histograms: building histograms without looking at data. In Alex Delis, Christos Faloutsos, and Shahram Ghandeharizadeh, editors, *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data: SIGMOD '99, Philadelphia, PA, USA, June 1–3, 1999*, volume 28(2) of *SIGMOD Record (ACM Special Interest Group on Management of Data)*, pages 181–192, New York, NY 10036, USA, 1999. ACM Press.

[Bio90] Biologie. *Encyclopedia of Life Science*. to be checked, 1990.

[Bro90] Brockhaus. *Brockhaus*. to be checked, 1990.

[CBS95] T. Connolly, C. Begg, and A. Strachan. *Database Systems: A Practical Approach to Design, Implementation and Management.* Addison Wesley, 1995.

[CG93] Peter Corrigan and Mark Gurry. *ORACLE Performance Tuning*. O'Reilly & Associates, Inc., 103a Morris Street, Sebastopol, CA 95472, USA, Tel: +1 707 829 0515, and 90 Sherman Street, Cambridge, MA 02140, USA, Tel: +1 617 354 5800, September 1993.

[CGN02] Surajit Chaudhuri, Ashish Kumar Gupta, and Vivek Narasayya. Compressing SQL workloads. In Michael Franklin and Bonki Moon ans Anastassia Ailamaki, editors, *Proceedings of the 2002 ACM SIGMOD international conference on Management of da-*

ta (SIGMOD-02), pages 488–499, New York, June 3–6 2002. ACM Press.

[CN97] Surajit Chaudhuri and Vivek R. Narasayya. An efficient cost-driven index selection tool for Microsoft SQL server. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases*, pages 146–155, 1997.

[CN98a] Surajit Chaudhuri and Vivek Narasayya. AutoAdmin "what-if" index analysis utility. In Laura Haas and Ashutosh Tiwary, editors, *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data: June 1–4, 1998, Seattle, Washington, USA*, volume 27(2) of *SIGMOD Record (ACM Special Interest Group on Management of Data)*, pages 367–378, New York, NY 10036, USA, 1998. ACM Press.

[CN98b] Surajit Chaudhuri and Vivek Narasayya. Microsoft index turning wizard for SQL Server 7.0. In Laura Haas and Ashutosh Tiwary, editors, *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data: June 1–4, 1998, Seattle, Washington, USA*, volume 27(2) of *SIGMOD Record (ACM Special Interest Group on Management of Data)*, pages 553–554, New York, NY 10036, USA, 1998. ACM Press.

[CN00] S. Chaudhuri and V. Narasayya. Automating statistics management for query optimizers. In *16th International Conference on Data Engineering (ICDE' 00)*, pages 339–348, Washington - Brussels - Tokyo, March 2000. IEEE.

[DR99] Donko Donjerkovic and Raghu Ramakrishnan. Probabilistic optimization of top n queries. In Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, editors, *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 411–422. Morgan Kaufmann, 1999.

[Fla97] Miroslaw A. Flasza. Implementation von Geo-Funktionalität in einem erweiterbaren relationalen DBMS. Master's thesis, Humboldt Universität zu Berlin, 1997.

[FM99] Pedro Furtado and Henrique Madeira. Summary grids: Building accurate multidimensional histograms. In *Database Systems for Advanced Applications*, pages 187–194, 1999.

[Fre00] J.C. Freytag. Script Vorlesung Grundlagen von Datenbanksystemen, 2000.

[Fre02] J.C. Freytag. Script Vorlesung Datenbanksystem Implementation of Database Systems, 2002.

[Fri91] Æleen Frisch. *Essential System Administration*. O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA, 1991.

[GLSW93] Peter Gassner, Guy M. Lohman, K. Bernhard Schiefer, and Yun Wang. Query optimization in the ibm db2 family. *IEEE Data Eng. Bull.*, 16(4):4–18, 1993.

[Hub03] Frank Huber. Studienarbeit - Automatische Konfiguration von DBM Systemen, 2003.

[Hun98] Craig Hunt. *TCP/IP Network Administration*. O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA, second edition, 1998.

[Ioa03] Yannis E. Ioannidis. The history of histograms. In *VLDB 2003, Proceedings of 29th International Conference on Very Large Data Bases, September 9-12, 2003, Berlin, Germany*, 2003.

[JKM+98] H. V. Jagadish, Nick Koudas, S. Muthukrishnan, Viswanath Poosala, Kenneth C. Sevcik, and Torsten Suel. Optimal histograms with quality guarantees. In Ashish Gupta, Oded Shmueli, and Jennifer Widom, editors, *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998,*

New York City, New York, USA, pages 275–286. Morgan Kaufmann, 1998.

[JKNS99] H. V. Jagadish, Olga Kapitskaia, Raymond T. Ng, and Divesh Srivastava. Multi-dimensional substring selectivity estimation. In Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, editors, *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 387–398. Morgan Kaufmann, 1999.

[KM90] Yves Kodratoff and Ryszard S. Michalski, editors. *Machine Learning, an Artificial Intelligence approach*, volume 3. Morgan Kaufmann, San Mateo, California, 1990.

[KS91] H. F. Korth and A. Silberschatz. *Database System Concepts*. McGraw-Hill, 1991.

[KW99] Arnd Christian König and Gerhard Weikum. Combining histograms and parametric curve fitting for feedback-driven query result-size estimation. In Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, editors, *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 423–434. Morgan Kaufmann, 1999.

[LKC99] Ju-Hong Lee, Deok-Hwan Kim, and Chin-Wan Chung. Multi-dimensional selectivity estimation using compressed histogram information. In Alex Delis, Christos Faloutsos, and Shahram Ghandeharizadeh, editors, *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA*, pages 205–214. ACM Press, 1999.

[LL02] Guy M. Lohman and Sam S. Lightstone. Smart: Making db2 (more) autonomic. In *VLDB'02, Proceedings of 28th International Conference on Very Large Data Bases*, 2002.

[Lou90] Mike Loukides. *System Performance Tuning*. O'Reilly & Associates, Inc., 981 Chestnut Street, Newton, MA 02164, USA, 1990. A fine book for system administrators on how to fine tune your UNIX system(s) to do more work.

[Mec95] Bernhard Mechler. *Intelligente Informationssysteme*. Addison-Wesley, 1995, 1995.

[Med90] Medizin. *Roche Lexikon Medizin*. to be checked, 1990.

[Mie01] Rosemarie Mielke. *Psychologie des Lernens*. Kohlhammer, 2001, 2001.

[Mir03] Mircosoft. Mircosoft research web site, 2003.

[ML02] Volker Markl and Guy Lohman. Learning table access cardinalities with LEO. In Michael Franklin and Bonki Moon ans Anastassia Ailamaki, editors, *Proceedings of the 2002 ACM SIGMOD international conference on Management of data (SIGMOD-02)*, pages 613–613, New York, June 3–6 2002. ACM Press.

[Nil96] Nils J. Nilsson. Introduction to machine learning, 1996.

[Ore86] Jack A. Orenstein. Spatial query processing in an object-oriented database system. In Carlo Zaniolo, editor, *Proceedings of the 1986 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 28-30, 1986*, pages 326–336. ACM Press, 1986.

[Ore89] Jack A. Orenstein. Strategies for optimizing the use of redundancy in spatial databases. In Alejandro P. Buchmann, Oliver Günther, Terence R. Smith, and Y.-F. Wang, editors, *Design and Implementation of Large Spatial Databases, First Symposium SSD'89, San-*

ta Barbara, California, July 17/18, 1989, Proceedings, volume 409 of *Lecture Notes in Computer Science*, pages 115–134. Springer, 1989.

[Pae90] Paedagogik. *dtv-Wörterbuch Pädagogik*. to be checked, 1990.

[Phi90] Philosophie. *Routledge Encyclopedia of Philosophy*. Routledge,to be checked, 1990.

[Qui93] J.Ross Quinlan. C4.5: Programs for machine learning, 1993.

[RZML02] Jun Rao, Chun Zhang, Nimrod Megiddo, and Guy Lohman. Automating physical database design in a parallel database. In Michael Franklin and Bonki Moon ans Anastassia Ailamaki, editors, *Proceedings of the 2002 ACM SIGMOD international conference on Management of data (SIGMOD-02)*, pages 558–569, New York, June 3–6 2002. ACM Press.

[SCF+86] Peter M. Schwarz, W. Chang, Johann Christoph Freytag, Guy M. Lohman, John McPherson, C. Mohan, and Hamid Pirahesh. Extensibility in the starburst database system. In Klaus R. Dittrich and Umeshwar Dayal, editors, *1986 International Workshop on Object-Oriented Database Systems, September 23-26, 1986, Asilomar Conference Center, Pacific Grove, California, USA, Proceedings*, pages 85–92. IEEE Computer Society, 1986.

[SLMK01] Michael Stillger, Guy Lohman, Volker Markl, and Mokhtar Kandil. LEO - DB2's learning optimizer. In *Proceedings of the 27th International Conference on Very Large Data Bases(VLDB '01)*, pages 19–28, Orlando, September 2001. Morgan Kaufmann.

[SSF01] Michael Stillger, Dieter Scheffner, and Johann-Christoph Freytag. A communication infrastructure for a distributed RDBMS (research note). *Lecture Notes in Computer Science*, 1900:445–??, 2001.

[Sta02] Werner Stangl. Lerndefinitionen, 2002. http://www.stangl-taller.at/ARBEITSBLAETTER/LERNEN/Lerndefinitionen.shtml.

[Sti00] Michael Stillger. *AQuES Ein flexibles, verteiltes System zur Optimierung und Auswertung relationaler Anfragen*. PhD thesis, Humboldt Universität zu Berlin, 2000.

[Tan87] Steven L. Tanimoto. *The Elements of Artificial Intelligence*. Computer Science Press, Rockville, Maryland, 1987.

[TO02] Michael Trimmel and Irina Onz. Lernen und Lerntheorien, 2002. http://mailbox.univie.ac.at/~ trimmem2/techpsych_ws2001-2002/onz.pdf.

[TPC04] Transaction Processing Performance Council TPC. TPC-H Benchmarks Version 2.1.8.1, 2004. http://www.tpc.org.

[VZZ+00] G. Valentin, M. Zuliani, D. Zilio, G. Lohman, and A. Skelley. DB2 advisor: An optimizer smart enough to recommend its own indexes. In *16th International Conference on Data Engineering (ICDE' 00)*, pages 101–110, Washington - Brussels - Tokyo, March 2000. IEEE.

[WVI97] Min Wang, Jeffrey Scott Vitter, and Balakrishna R. Iyer. Selectivity estimation in the presence of alphanumeric correlations. In Alex Gray and Per-Åke Larson, editors, *Proceedings of the Thirteenth International Conference on Data Engineering, April 7-11, 1997 Birmingham U.K*, pages 169–180. IEEE Computer Society, 1997.

[ZLLL01] Daniel Zilio, Sam Lightstone, Kelly Lyons, and Guy Lohman. Self-Managing technology in IBM DB2 universal database. In Henrique Paques, Ling Liu, and David Grossman, editors, *Proceedings of the Tenth International Conference on Information and Knowledge Management (CIKM-01)*, pages 541–543, New York, November 5–10 2001. ACM Press.

Abbildungsverzeichnis

Tabellenverzeichnis

Liste der Algorithmen