

Otto-von-Guericke-Universität Magdeburg



Fakultät für Informatik
Institut für Technische und Betriebliche Informationssysteme

Diplomarbeit

Self-Tuning-Konzepte zur Verwaltung von Bitmap-Index-Konfigurationen

Verfasser:

Andreas Lübcke

12. April 2007

Betreuer:

Prof. Dr. rer. nat. habil. Gunter Saake,
Dr.-Ing. Eike Schallehn

Universität Magdeburg
Fakultät für Informatik
Postfach 4120, D-39016 Magdeburg
Germany

Lübcke, Andreas:

*Self-Tuning-Konzepte zur Verwaltung von
Bitmap-Index-Konfigurationen*

Diplomarbeit, Otto-von-Guericke-Universität
Magdeburg, 2007.

Danksagung

Ich möchte diesen kurzen Abschnitt nutzen, um mich für die gute Zusammenarbeit während der Bearbeitung des Diplomthemas zu bedanken. Besonderer Dank gilt meinen Betreuern Gunter Saake und Eike Schallehn sowohl für die sehr gute fachliche als auch menschliche Betreuung, sowie ich Georg Paul meine Anerkennung für die fachliche und schnelle Arbeit als Gutachter aussprechen möchte. Weiterhin möchte ich mich bei Martin Kuhleemann, Kerstin Gießwein, Fred Kreuzmann und Steffem Thorhauer für die freundliche Hilfe bei Fragen und technischen Problemen bedanken. Allerdings möchte ich auch alle anderen Mitarbeitern des Lehrstuhles Datenbanken am Institut für Technische und Betriebliche Informationssysteme auszeichnen, die mir mit Rat und Tat zur Seite standen. Vielen Dank an alle, die diese Arbeit in anderer Form unterstützt haben, für die sehr gute und freundliche Zusammenarbeit.

Inhaltsverzeichnis

Inhaltsverzeichnis	iii
Abbildungsverzeichnis	v
Tabellenverzeichnis	vii
Verzeichnis der Abkürzungen	ix
1 Einleitung	1
1.1 Motivation	1
1.2 Zielsetzung	2
1.3 Ausführliche Gliederung	2
2 Grundlagen des Index-Tuning	5
2.1 Index-Strukturen	5
2.2 Bitmap-Indexe	6
2.3 B/B ⁺ -Baum Index-Strukturen	7
2.4 Join-Indexe und weitere Indexverwendungen	9
2.5 Anwendungsgebiet	10
2.6 Verwendete Techniken	11
3 Self-Tuning von Index-Strukturen	13
3.1 Kosten-/Nutzenbetrachtungen von Indexkandidaten	13
3.2 Regelkreis Index-Self-Tuning	14
3.3 MAPE von IBM	15
3.4 Physical Design Alerte von Microsoft	16

4	Konzept für das Self-Tuning von Bitmap-Indexen	19
4.1	Herangehensweise und Entwurf eines Kostenmodells	19
4.2	Profit von Index-Empfehlungen	21
4.3	Anfrageanalyse für Bitmap-Index-Empfehlungen	23
4.4	Entscheidungsmodell für Bitmap-Index Selektion	27
4.5	Funktionsweise und Einbettung	31
5	Evaluierung	35
5.1	Zugriffs- und Such-Bitmap-Index gegenüber B/B ⁺ -Baum	35
5.2	Analyse von Bitmap-Join-Indexen und STAR-Joins	40
5.3	Anwendung und Ergebnis	42
6	Zusammenfassung	45
A	Anhang	49
A.1	Beispiel für Zugriffs- und Such-Bitmap-Indexe	49
A.2	Beispiel für Bitmap-Join-Indexe und STAR-Joins	56
	Literaturverzeichnis	67

Abbildungsverzeichnis

2.1	Beispiel für einen Bitmap-Index [SHS05, Seite 334]	7
2.2	Beispielhafte Darstellung eines B-Baumes	8
2.3	Indexgraph für Multiindex [SHS05, Seite 261-268]	10
2.4	Indexgraph für Verbundindex [SHS05, Seite 261-268]	10
2.5	Indexgraph für Pfadindex [SHS05, Seite 261-268]	10
2.6	Beispiel für ein STAR-Schema	11
3.1	Regelkreis Index-Self-Tuning	15
3.2	Darstellung des Zykluses des MAPE-Ansatzes	16
3.3	Monitor-Diagnose-Tune-Zyklus des Physical Design Problem [BC06]	17
4.1	STAR-Schema für Oracle-Bitmap-Join-Index	27
5.1	Verwendetes Schema für die Cube-Berechnung	36
5.2	Nach empfundenes Beispiel für Bitmap-Join-Indexe	41

Tabellenverzeichnis

5.1	CPU Costs der einzelnen Indexvarianten	38
5.2	Speicherplatzbedarf von Index-Strukturen	39
5.3	Verarbeitungszeiten des verwendeten Skriptes und einer Testanfrage . . .	40

Verzeichnis der Abkürzungen

- AWK** Attributwertkardinalität
- DDL** Data Description Language
- DML** Data Manipulation Language
- DBA** Datenbankadministrator
- DBMS** Datenbankmanagementsystem
- DBS** Datenbanksystem
- DWH** Data-Warehouse
- GHz** GigaHertz
- ISP** Index Selection Problem
- KB** KiloByte
- MAPE** Monitor Analyze Plan Execute
- MB** MegaByte
- OLAP** Online Analytical Processing
- ROLAP** Relational Online Analytical Processing
- SQL** Structured Query Language
- TID** Tupelidentifikator

Kapitel 1

Einleitung

In dieser Arbeit sollen Erkenntnisse, die im Rahmen der wissenschaftlichen Tätigkeit am Institut für Technische und Betriebliche Informationssysteme der Fakultät für Informatik an der Otto-von-Guericke-Universität Magdeburg erlangt wurden, vorgestellt und wissenschaftlich erörtert werden.

Insbesondere werden dabei das Self-Tuning von Index-Konfigurationen innerhalb eines Datenbankmanagementsystems (**DBMS**) und im Speziellen die Anwendung für Bitmap-Index-Strukturen betrachtet.

1.1 Motivation

Das Thema der Self-Tuning-Konzepte für Bitmap-Index-Konfigurationen wurde durch bisherige Projekte im Bereich des Datenbankentwurfs¹ und der physischen Datenbankoptimierung² an Data-Warehouse-ähnlichen Analyseplattformen motiviert.

Da diese Systeme durch ihre weit gefächerten Anwendungsgebiete in der Wirtschaft sehr verbreitet sind, und somit eine immer wichtigere Rolle für den wirtschaftlichen Erfolg einer Firma übernehmen, ist die Weiterentwicklung dieser Systeme und unterstützender Konzepte von großer wirtschaftlicher Bedeutung.

Eine Optimierung dieser Systeme kann wie auch in jedem anderen Datenbanksystem (**DBS**) mit Hilfe von unterschiedlichsten Ansätzen und Mechanismen geschehen, welche für Datenbanken und besonders für Data-Warehouses (**DWH**) von Bedeutung sind. Denn die Datenbestände innerhalb der Systeme wachsen ständig, und es sich zumeist um zeitkritische Operationen handelt. Dies sind Gründe dafür, daß die stetige Optimierung zur Laufzeit der Systeme (keine Offline-Zeit für Optimierung, das heißt Online-Optimierung) immer mehr in die Betrachtungen der aktuellen Forschung einbezogen werden muß. Das geschieht bereits seit einigen Jahren, aber trotz zahlreicher Fortschritte in aktuellen Versionen der DBMS-Hersteller sind die Forschungen noch nicht an einem Punkt angekommen, an dem ein autonomes Tuning möglich ist.

Inhaltlich wird sich diese Arbeit daher auf Ansätze für die Optimierung von Datenbanken im Bereich des Self-Tunings beschränken, um ausgehend von bisherigen Konzepten speziell auf neue Vorschläge für Bitmap-Index-Konfigurationen zu schließen. Bisher fanden

¹Gen-Datenbank-Entwurf am Leibniz-Institut für Pflanzengenetik und Kulturpflanzenforschung

²Im Rahmen der Studienarbeit zum Praktikum bei der DaimlerChrysler AG

diese wenig oder gar keine Beachtung in kommerziellen Systemen, obwohl sich gerade im Bereich des Data-Warehouses sehr gute Anwendungsmöglichkeiten für diese Art von Index-Strukturen finden lassen. Und dadurch das Anwendungs- und Optimierungsspektrum gegenüber aktuellen Systemen und Konzepten erweitert wird.

1.2 Zielsetzung

Während der wissenschaftlichen Tätigkeit war es die Aufgabe Konzepte für einen Index- oder Design Advisor [ACK⁺04, ZRL⁺04] zu entwerfen, die eine performante Empfehlung und Erstellung von Bitmap-Index-Konfigurationen ermöglichen. Diese sollen auf den bisherigen Möglichkeiten solcher Advisoren aufbauen, und somit aktuelle Ansätze für eine derartige Erweiterung nutzen.

Zuerst soll eine Einzellösung für die neue Form von Index-Konfigurationen gefunden werden, um die Vor- und Nachteile gegenüber den bisherigen Strukturen aufzeigen zu können. Auf diese Art und Weise wird das mögliche Anwendungsgebiet für zukünftige Arbeiten abgesteckt, wodurch eine genaue Abwägung getroffen werden kann, für welche Index-Konfigurationen sich entschieden werden soll. Das sich daraus ergebende Teilergebnis sieht ein eigenständiges Kostenmodell und dessen Verwendung für die Nutzung der Bitmap-Index-Konfigurationen vor.

Diese Erweiterung und das zugehörige Abschätzungsmodell werden aber die bisherigen Standards nicht ersetzen, sondern werden in einem dynamischen und autonomen Kontext verwendet. Denn wie jede andere Indexstruktur werden auch Bitmap-Indexe [CI98] nicht alle möglichen Anwendungsfälle in einer zufrieden stellenden Art und Weise abdecken können, daher sollen sie als Unterstützung für bisherige Standards genutzt werden. Dies führt zu einer Erweiterung des bisherigen Spektrums an Optimierungsmöglichkeiten für Index- oder Design Advisor.

Ein wünschenswertes Ziel wäre es daher, eine Mischung aus verschiedenen Index-Konfigurationen zu erreichen, um so für die verschiedenen Anwendungsgebiete innerhalb eines Systems eine performantere Lösung zu finden als es bisher möglich war. Die Realisierung könnte auf verschiedene Arten geschehen, da eine Mischung von verschiedenen Index-Strukturen innerhalb eines Index-Pools (Bereich in dem Indexe erstellt und verwaltet werden) nicht möglich ist, und dadurch die Kopplung der Index-Strukturen autonom und unabhängig voneinander stattfinden muß.

1.3 Ausführliche Gliederung

Dieser Abschnitt wird den Aufbau und die Struktur der Arbeit mit Hilfe eines kurzen Überblicks vorstellen.

Im folgenden zweiten Kapitel werden die Grundlagen erläutert, die für die wissenschaftliche Ausarbeitung notwendig waren. Es wird auf grundlegende Dinge aus dem Bereich der Index-Strukturen sowie des Self-Tunings eingegangen, die für das weitere Verständnis dieser Arbeit von fundamentaler Bedeutung sind. Weiterhin werden die Unterschiede zu bisherigen Ansätzen und die möglichen Anwendungsbereiche des neuen Ansatzes erläutert.

Das dritte Kapitel wird einen genaueren Einblick in die Thematik geben, und die Heran-

gehensweise, die Konzeptentwicklung an sich und die entstandenen Modelle beschreiben. Es folgt eine Abgrenzung zu bisherigen Anwendungsszenarien und deren Konzepten, wodurch die Grenzen und Anforderungen dieses Ansatzes aufgezeigt werden sollen.

Im vierten Kapitel werden die Vorschläge und deren Auswirkungen beschrieben. Dabei wird auf die verschiedenen Ansätze und Möglichkeiten eingegangen, und diese miteinander verglichen und analysiert. Die Evaluierung der Ergebnisse und des Gesamtkonzeptes sollen die Möglichkeiten bestätigen und beenden dieses Kapitel.

Abschließend ist das fünfte Kapitel der zusammenfassenden Darstellung der Ergebnisse und den Erkenntnissen dieser Arbeit gewidmet. Es beinhaltet einen kurzen Ausblick, und zeigt mögliche Aufgaben für die Zukunft auf.

Der sich am Ende dieser Arbeit befindende Anhang enthält, die für die Evaluierung verwendeten, Skripte zur Erstellung der gewählten Beispielszenarien.

Kapitel 2

Grundlagen des Index-Tuning

Dieses Kapitel wird grundlegende Begriffe und Sachverhalte, die für das weitere Verständnis der Arbeit wichtig sind, vorstellen. Weiterhin werden die theoretischen Grundlagen aus dem Bereich Datenbanken in Bezug auf diese Arbeit dargelegt, wobei dann speziell auf das nötige Wissen für die nachfolgenden Analysen und Schlußfolgerungen eingegangen wird.

2.1 Index-Strukturen

Dieser Abschnitt wird den Begriff des Indexes in Bezug auf Datenbanken allgemein erläutern, und dabei auf die verschiedenen Formen und deren Unterschiede innerhalb dieses Teilbereiches der Datenbankmanagementsysteme eingehen.

Allgemein sind Indexe Zugriffsstrukturen, die den Zugriff auf gewünschte Daten beschleunigen sollen, da durch sie nicht der gesamte Datenbestand des gewünschten Abschnitts (zumeist Relationen) [SHS00] durchsucht werden muß. Dabei werden Indexe über ein ausgewähltes Attribut erstellt, um eine effektive Suche über den Wertebereich dieses Attributes zu erreichen, denn im Vergleich zur Gesamtgröße ist die angefragte Menge meist sehr gering. Daraus folgt dann, daß keine nicht-relevanten Datensätze mit durchsucht werden müssen.

Durch die Wahl des Attributes ergibt sich auch die erste Unterscheidung von Index-Strukturen, denn es gibt Primärindexe, die über das Primärattribut einer Relation angelegt werden und damit einelementig sind, sowie Sekundärindexe, die über jedes beliebige andere Attribut oder Menge von Attributen einer Relation angelegt werden können. Die Menge der Attribute, die durch einen Index abgedeckt werden, legen dessen Dimensionalität fest. Dabei muß das Primärattribut Schlüsseigenschaften³ besitzen, und dessen Index kann dabei die interne Dateioorganisation und Sortierung der Speicherstruktur ausnutzen. Bei Sekundärindexten dagegen müssen weder die Schlüsseigenschaften erfüllt sein, noch können sie die Organisationsstruktur ausnutzen, da diese durch die Sortierung des Primärattributes vorgegeben wird. Dies führt zu einem weiteren Unterscheidungsmerkmal, denn durch die Nutzung der internen Darstellung der Relation ist ein Primärindex geclustert, das heißt in gleicher Form sortiert wie die interne Relation, und ein Sekundärindex gerade nicht.

³Eindeutig und identifizierend

Weiterhin können die Indexe in unterschiedlicher Form belegt sein, was ebenfalls mit der Art des Indexes zusammenhängt. Der Unterschied dabei liegt zwischen dicht- und dünnbesetzten Indexen, wobei nur Primärindexe durch ihre obigen Eigenschaften dünnbesetzt sein dürfen. Dagegen sind Sekundärindexe immer dichtbesetzt und nicht-geclustert, was sich aus den bisherigen Betrachtungen schließen läßt. Wiederum ist jeder dünnbesetzte Index durch die bisherigen Einschränkungen immer ein geclustertes Index, dies gilt aber nicht umgekehrt [SHS05, 50,144-151].

Dadurch sind die beiden Klassen von Indexen eindeutig voneinander unterschiedbar, und in ihren Eigenschaften klar definiert.

2.2 Bitmap-Indexe

Aufbauend auf den vorherigen Abschnitt sollen sich die folgenden Betrachtungen den während der wissenschaftlichen Tätigkeit genauer betrachteten Bitmap-Index-Strukturen widmen, um das weitere Verständnis für die späteren Ansätze zu fördern.

Um erläutern zu können, wie ein Bitmap-Index arbeitet, muß zuerst der Begriff der Tupelidentifikators (**TID**) geklärt werden. Die Liste von Tupelidentifikatoren [SHS05, Seite 108] enthält die Seitennummer des jeweiligen Tupels⁴ und dessen Offsetadresse, welche den Speicherort innerhalb einer Seite angibt. Mit Hilfe dieses Konzeptes können die Datensätze gefunden werden, ohne alle Datenseiten durchsuchen zu müssen. Ein weiterer Vorteil dieser indirekten Adressierung ist es, daß eine Änderung der Position des gewünschten Tupels nur eine Veränderung des lokalen (internen, innerhalb der Seite) Zeigers benötigt. Denn die Adressierung, auf die das Datenbankmanagementsystem zugreift, bleibt stabil, solange keine Reorganisation notwendig ist. Durch einen festgelegten Grad der Verzeigerung wird diese angestoßen, um eine ineffiziente Zeigerverwaltung zu verhindern.

In einem Bitmap-Index werden diese Tupelidentifikatoren eines Schlüsselwertes durch eine Menge von Bitvektoren ersetzt. Im Beispiel (**Abbildung 2.1**) soll der Bestellstatus⁵ indiziert werden, und dadurch wird der Aufbau von Bitmap-Indexen verdeutlicht. Diese Art der Speicherung in Form von Bitvektoren benötigt wesentlich weniger Speicherplatz als andere Index-Strukturen wie zum Beispiel der B/B⁺-Baum, und erfordert bei bekannten Attributwerten einen linearen Aufwand für die Erstellung. Dadurch bleibt offen wie der Platzbedarf und die Erstellungskosten gegenüber anderen Index-Strukturen gewertet werden kann. Weiterhin wird bei Bitmap-Indexen jede Dimension getrennt von der Anderen gespeichert, sowie ein Bitvektor für jeden möglichen Wert des Attributes angelegt. Besonders effizient ist ein Bitmap-Index, wenn es nur wenig mögliche Schlüsselwerte gibt, die durch den Index dargestellt werden müssen. Weiterhin können die Bitvektoren sehr leicht logisch miteinander verknüpft werden, um so komplexe Selektionsbedingungen auszuwerten. Zusätzlich sind diese Indexe im Gegensatz zu Baumverfahren relativ unempfindlich gegen eine höhere Zahl von Dimensionen, und können so Anfragen leichter unterstützen, die auf einige Dimensionen beschränkt sind.

Aber auch diese Art der Indexstruktur hat ihre Nachteile, denn durch ihre Struktur sind Änderungsoperationen sehr aufwendig, da für sie Matrixmodifikationen nötig werden. Durch den hohen Aufwand erhöht sich die Wahrscheinlichkeit von Sperrkonflikten, das

⁴Der angefragte Datensatz ist ein oder eine Menge von Tupeln

⁵B = in Bearbeitung, F = fertig, O = offen

TID	B	F	O
rowid ₁	0	0	1
rowid ₂	0	1	0
rowid ₃	0	0	1
rowid ₄	1	0	0
rowid ₅	1	0	0
rowid ₆	0	1	0
rowid ₇	0	0	1
⋮	⋮	⋮	⋮

Abbildung 2.1: Beispiel für einen Bitmap-Index [SHS05, Seite 334]

bedeutet Datensätze werden für Zugriffe gesperrt, was wiederum zum Blockieren des Datenbanksystems führen kann. Zusätzlich würde jedes Löschen eines Tupels zu einer Reorganisation des Indexes führen, dies kann aber durch eine Hilfsspalte zur Markierung dieser Tupel umgangen werden. Allerdings verhindert diese Methode nicht, daß eine Reorganisation zu einem späteren Zeitpunkt notwendig wird.

Weiterhin ergeben sich, aufgrund der in diesem Abschnitt erläuterten Struktur von Bitmap-Indexen, neue Gesichtspunkte, die für diese Ansätze Beachtung finden müssen. Als Erstes ist es wichtig zu beachten, daß nur die Prädikate der WHERE-Klausel in Form $A = const$ einer Relation $r(R)$ relevant sind. Dabei muß die Bedingung für die Spalte $A \in R$ eine Attributwertkardinalität (**AWK**) $card(dom(A))/card(r(R)) < maxSelectivity$, das heißt die Anzahl möglicher Werte von A ist gering gegenüber der Größe der Relation, erfüllt werden. Weiter stellt sich die Frage, wie mit der Möglichkeit von Mehrkomponenten-Bitmap-Indexen in späteren Betrachtungen umgegangen werden soll, da diese Art der Indexe Einschränkungen für die Verwendung und Suche von Bitmap-Index-Konfigurationen bewirken können.

Der Schluß dieser Betrachtungen ist, daß der Aufwand von Änderungsoperationen und den weiteren Nachteilen gegenüber den Vorteilen von Bitmap-Index-Strukturen genau abgewogen werden muß.

2.3 B/B⁺-Baum Index-Strukturen

Im Zuge der späteren vergleichenden Betrachtungen zu bisherigen Index-Strukturen, soll in diesem Abschnitt das Konzept des B-Baumes erläutert werden. Aufgrund der Struktur von Bäumen ist nahe liegend, daß es sich hier um eine eindimensionale Indexstruktur handelt.

Ein B-Baum ist ein Suchbaum, der im Gegensatz zu herkömmlichen binären Suchbäumen mehrere Schlüsselwerte je Knoten zuläßt. Die Anzahl der Werte wird durch die so genannte Ordnung (im Folgenden als m bezeichnet) des B-Baumes bestimmt, wodurch jeder Knoten außer der Wurzel selbst dann mindestens m aber alle maximal $2 * m$ Schlüsselwerte enthält. Im Regelfall wird der Wert m so angepaßt, daß die maximale Größe der

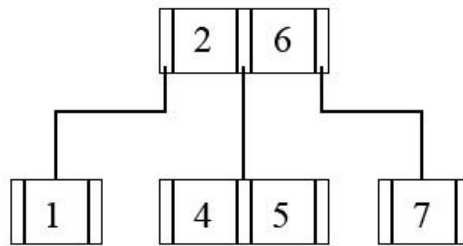


Abbildung 2.2: Beispielhafte Darstellung eines B-Baumes

Datensätze der Blockgröße⁶ oder einem Vielfachen des Datenträgers entspricht. Bei einem Unter- oder Überlauf dieser Werte muß der B-Baum reorganisiert werden. Dabei muß beachtet werden, daß ein B-Baum immer ausgeglichen sein muß. Diese Eigenschaft hat den Effekt, daß alle äußeren Knoten auf der gleichen Höhe h liegen, und sie damit alle den gleichen Abstand zur Wurzel besitzen. Allerdings kann diese Eigenschaft des B-Baumes durch das Zusammenfügen oder Trennen von Knoten eine Kette von Reorganisationsoperationen auslösen, um wieder zu einer ausgeglichenen Struktur zu kommen [BM72]. Die verschiedenen Kriterien des B-Baumes (**Abbildung 2.2**) führen zu mehreren für Index-Strukturen interessanten Eigenschaften, denn bei n Datensätzen werden nur $\log_m n$ Knotenzugriffe benötigt, was einen wesentlichen Gewinn gegenüber der linearen Suche (auch Einfügen und Löschen) bedeutet. Aber diese Eigenschaft hat ebenfalls ihre Grenzen, denn wird die Menge der angefragten Datensätze zu groß, dann ist die Verwendung des B-Baumes durch die mehrfache Suche in ihm ineffektiver als ein lineares Durchsuchen der Daten. Heuristiken und Analysen legen den Schwellwert für einen so genannten Full Table Scan auf circa zehn Prozent der gesamten Daten innerhalb des B-Baumes fest. Weiterhin kann durch die obigen Forderungen zu Gunsten der Performance eine recht gute Speicherplatzausnutzung von annähernd fünfzig Prozent erreicht werden.

Des Weiteren wurde das ursprüngliche Konzept des B-Baumes um zwei wesentliche Ansätze erweitert. Als erstes ist der B⁺-Baum [CDF⁺01] zu nennen, der sich dahingehend von den ersten Vorschlägen unterscheidet, daß die inneren Knoten keine Daten enthalten. Das bedeutet, daß nur noch die Blattknoten direkt auf die realen Datensätze in der Hauptdatei verweisen, und die inneren Knoten nur noch Verzeigerungen enthalten, um so zum richtigen Blattknoten zu gelangen. Dadurch können diese Daten dann in der Ordnung des Indexes sequentiell gelesen werden, wenn dies nötig ist. Dies bezieht sich auf die Problematik zu vieler angefragter Daten des Indexes, um so eine performantere Lösung bei Anfragen auf Bereiche von Daten gegenüber dem Full Table Scan zu erreichen. Dieser Ansatz zählt zu den meist verwendeten Index-Strukturen überhaupt.

Der zweite Ansatz des B*-Baum dagegen zielt auf eine andere Problematik. Denn mit ihm wurde die Reorganisation des B-Baumes so verändert, daß bei einem Überlauf eines Knotens dieser nicht aufgeteilt wird. Es wird versucht den Überlauf durch die Verteilung auf benachbarte nicht voll belegte Seiten zu lösen, wodurch eine häufige Aufspaltung von Knoten reduziert wird. Falls dies nicht möglich ist, werden zwei volle Knoten in drei Knoten aufgeteilt, dadurch wird statt circa fünfzig Prozent Speicherplatzausnutzung ein Wert von ungefähr sechsundsechzig Prozent erreicht.

⁶Kleinste Dateneinheit eines Datenträgers

2.4 Join-Indexe und weitere Indexverwendungen

Nach den vorherigen Betrachtungen soll an dieser Stelle beschrieben werden, welche Möglichkeiten bestehen Indexe für die Optimierung in einem Datenbankmanagementsystem zu verwenden.

Die erste Verwendungsmöglichkeit ergibt sich trivial durch die Beschreibungen aus **Abschnitt 2.1**, denn zuerst ergibt sich dadurch ein schnellerer Zugriff auf angefragte Daten einer Relation. Dafür kann idealer Weise ein Index verwendet werden, solange die angefragte Menge von Tupel nicht zu groß wird gegenüber der Größe der Relation. Da ab einer gewissen Grenze, die vom Typ des Indexes abhängt, ein Indextdurchlauf (auch Index Scan genannt) ineffizienter wird als ein Durchlauf der gesamten Relation (wird auch als Table Scan bezeichnet) in einer bestimmten Sortierreihenfolge. Denn so muß jedes Tupel nur einmal angefragt werden, dagegen kann es passieren, daß bei großen Anfragemenge über einen Index das gleiche Tupel mehrmalig angefragt werden muß, um das Anfragergebnis zu erhalten. Also besteht schon an dieser Stelle der Bedarf von Entscheidungsmodellen für die verschiedenen Indextypen, dabei werden nur Anfragen (Lesetransaktionen) an eine Relation betrachtet.

Zudem kann die immer existierende Sortierung innerhalb eines Indexes ausgenutzt werden, da auf diese Art und Weise auch verschiedene, häufig verwendete Sortierungen genutzt werden können. Dies ist insofern trotz der Bildung von mehreren Indexen mit gleichem Inhalt (Redundanz) sinnvoll, wenn dadurch die Umberechnungen von Sortierungen bei gestellten Anfragen entfallen, die eine dem Index nicht entsprechende Sortierung benötigen. Wenn dieser Fall oft genug (abhängig vom speziellen Anwendungsfall) eintritt, dann ist es sinnvoll den Nachteil der Redundanz in Kauf zu nehmen, denn Sortierungen sind sehr aufwendige Operationen während einer Anfrage, die so umgangen werden können. Man betrachte zur Verdeutlichung den Unterschied zwischen Nested-Loop-Join (nutzt Sortierung) und Merge-Join (Sortierung vor Join), um die Problematik der Sortierung zu verdeutlichen [SHS05, Seite 16,351-357].

Für die Verwendung von Join-Indexen muß zuerst der Begriff des Multiindex dargelegt werden. Dieser unterstützt den Zugriff über einen Pfadzugriff, welcher durch die Beziehung von Komponenten (in diesem Beispiel sind es Relationen) beschrieben wird. Die Realisierung erfolgt durch binäre Indexdateien, die eine Verbindung von der n -ten zur $n-1$ -ten Komponente herstellt (**Abbildung 2.3**). Dadurch ist ein Zugriff von einer Seite über die verschiedenen Beziehungen zur Anderen möglich, was gleichzeitig ein Nachteil dieser Indexform ist. Weiterhin ist es für Anfragen über mehr als eine Komponente, um zum Beispiel vom Lektor auf die beim Verlag veröffentlichten Bücher zu schließen, nötig mehrere Indexdateien miteinander zu verknüpfen.

Ein Verbundindex (Join-Index) ist nun gerade ein symmetrischer Multiindex, der durch jeweils zwei Indexdateien den beidseitigen Zugriff zwischen zwei Komponenten ermöglicht. Dies ist in **Abbildung 2.4** dargestellt, und zeigt wie zu den Beziehungen je zwei Indexzugriffsdateien zur Verfügung gestellt werden. Auch hier bestehen die Nachteile von Multiindexen weiter, da es sich eigentlich nur um einen Multiindex und dessen gespiegelte Form handelt.

Um diese Nachteile zu umgehen, gibt es den Vorschlag der geschachtelten Indexe (Nested-Index). Diese ermöglichen, durch eine einzige Indexdatei, den Zugriff von der n -ten Komponente auf die Erste, aber auch nur auf diese. Denn die Umsetzung erfolgt ohne Verwendung von Teilpfaden. Darauf wiederum folgt ein verallgemeinerter Vorschlag, der so

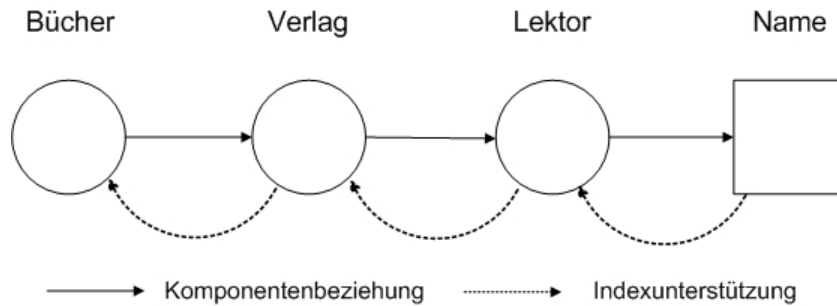


Abbildung 2.3: Indexgraph für Multiindex [SHS05, Seite 261-268]

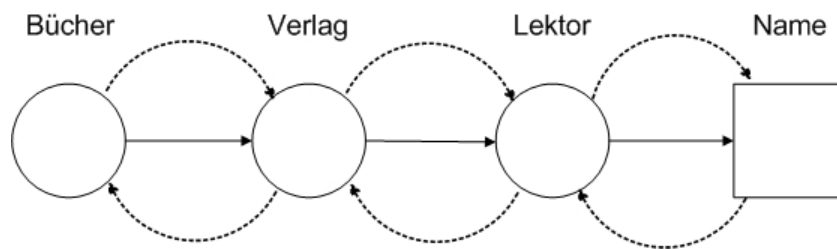


Abbildung 2.4: Indexgraph für Verbundindex [SHS05, Seite 261-268]

genannte Pfadindex. Dieser erweitert den Ansatz des geschachtelten Indexes so, daß ein Zugriff von der n -ten Komponente zu jedem ihrer Vorgänger möglich ist. Dies erfolgt wieder nur auf direktem Wege wie es **Abbildung 2.5** zeigt, da auch hier keine Teilpfade Verwendung finden [SHS05, Seite 261-268]. Der Bezug zu den Themen Bereichsanfragen und Nachbarschaftssuche soll an dieser Stelle vernachlässigt werden, da sie für die weiteren Ausführungen nicht von Bedeutung sind.

2.5 Anwendungsgebiet

Aufgrund der bisher vorgestellten Komplexität der Problemstellung wird das Anwendungsgebiet an dieser Stelle eingeschränkt, um so das Augenmerk auf die betrachteten Vorschläge zu beziehen. Neben den in **Abschnitt 2.4** vorgestellten Bereichen wie Zugriffs- und Suchoptimierung soll in diesem Abschnitt zusätzlich der spezielle Bezug zum Modell des Data-Warehouses hergestellt werden.

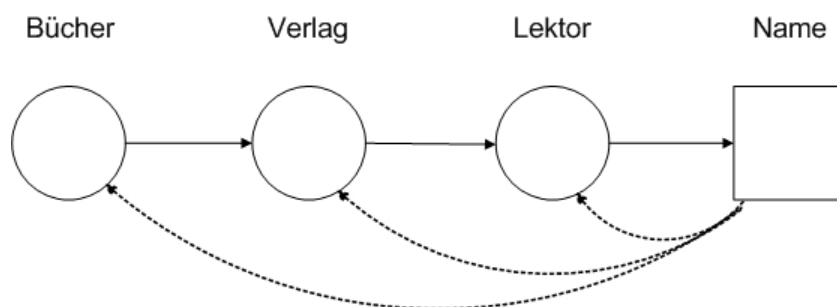


Abbildung 2.5: Indexgraph für Pfadindex [SHS05, Seite 261-268]

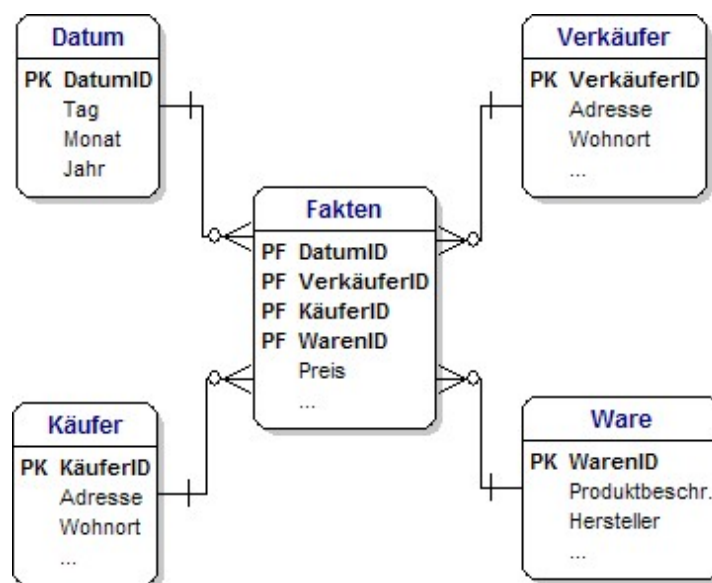


Abbildung 2.6: Beispiel für ein STAR-Schema

Dazu sollen zunächst die Konzepte STAR-Schema und STAR-Join [Leh03, Seite 85-92] vorgestellt werden. Das STAR-Schema ist ein Modell zur multidimensionalen Darstellung von Daten in relationalen Datenbanken, und unterliegt damit den Prinzipien des ROLAP-Ansatzes [Leh03, Seite 55,83].

Im STAR-Schema (Stern-Schema) selbst werden diese Daten mit Hilfe einer so genannten Faktentabelle und separaten Relationen für die Dimensionen dargestellt **Abbildung 2.6**. Dabei setzt sich der Schlüssel der Faktentabelle aus den Schlüsseln der Dimensionstabellen zusammen, und durch die Fremdschlüsselbedingungen wird sichergestellt, daß die Fakten tatsächlichen Dimensionswerten zugeordnet sind. Weiterhin existiert zu jeder Dimension nur eine Relation, die denormalisiert sein kann, weil die Hierarchien innerhalb der Dimensionen nicht abgebildet werden, und diese dadurch zu transitiven Abhängigkeiten führen. Allerdings ist damit auch die Möglichkeit gegeben, Redundanzen für die Anfrageoptimierung zu nutzen.

Der STAR-Join dagegen ist der Verbund über einen, im STAR-Schema modellierten, Datenbestand, bei dem die Relationen mittels Verbundindex miteinander verknüpft werden (siehe auch **Abschnitt 2.4**). Dabei werden diese vorberechneten Verbunde materialisiert [Val87]. Diese Art des Joins kann zusätzlich durch spezielle Verbund-Algorithmen und Zugriffsstrukturen (Indexe) unterstützt werden. Dies ist von hoher Bedeutung, da es sich um eine sehr häufige Operation in Anwendungen für Data-Warehouses handelt, denn STAR-Joins haben ein typisches Muster für diese Art von Anfragen.

Aufgrund der Struktur von Bitmap-Indexen sind sie in diesem Bereich besonders geeignet für die Umsetzung auf Dimensionstabellen, und damit ist es eine sinnvolle Überlegung diese ebenfalls für Joins und speziell den STAR-Join zu verwenden.

2.6 Verwendete Techniken

In diesem Abschnitt werden die verwendeten Techniken vorgestellt, die für die Ausarbeitung der verschiedenen Konzepte und Werte für die Beweisführung gewonnen wurden.

Zunächst ist die verwendete Software (Datenbankmanagementsystem) für den Datenbankserver zu nennen, dabei handelt es sich um die Version Oracle Database 10g Release 2. Weiterhin wurde für die Fernverwaltung des Datenbankmanagementsystems die Client Software aus dem Hause Oracle genutzt, dabei war die Version Oracle Database 10g Client Release 2 auf der Arbeitsstation installiert.

Für den Entwurf der verschiedenen Beispielszenarien und deren grafische Darstellung wurde eine Version des Design- und Entwicklungswerkzeug DeSign for Databases 3.4.1 von Datanamic verwendet. Des Weiteren konnte das, von Oracle zur Verfügung gestellte, Entwickler-Tool Oracle SQL Developer in der Version 1.1.2.25 genutzt werden. Dieses bot eine komfortable grafische Oberfläche für die Arbeit an der Datenbank selbst und den verwendeten Beispielszenarien.

Mit Hilfe der Entwicklungs- und Optimierungsmöglichkeiten dieser verschiedenen Softwareprodukte konnte die Entwicklung der verschiedenen Konzepte und Kostenbetrachtungen durchgeführt werden. Die Gewinnung der Werte für die Evaluierung wurden ebenfalls mit Hilfe der Statistik- und Analysefunktionen dieser Softwareprodukte ausgeführt.

Für die Erstellung der schriftlichen Ausarbeitung konnten Versionen von MiKTeX (2.5.2443) und TeXnicCenter (1 Beta 7.01) sowie der Adobe Reader 8.0 genutzt werden.

Kapitel 3

Self-Tuning von Index-Strukturen

Aufbauend auf die grundlegenden Darstellungen des vorherigen Kapitels wird dieses Kapitel speziell die Grundlagen aus dem Bereich des Self-Tunings erläutern. Zunächst finden allgemeine Betrachtungen zur Thematik statt, um diese für das spätere Verständnis nutzen zu können. Woraufhin einige Ansätze aus der aktuellen Forschung für die nachfolgenden Schlußfolgerungen vorgestellt werden.

3.1 Kosten-/Nutzenbetrachtungen von Indexkandidaten

Im Folgenden wird ein Einblick darüber gegeben, in welchem Bereich das so genannte Self-Tuning angesiedelt ist. Als Teilgebiet der Datenbankoptimierung ist es auch an dieser Stelle das Ziel eine möglichst performante Datenverarbeitung zu gewährleisten, und/oder diese weiter zu steigern. Weiterhin wird der Aufgabenbereich dahingehend spezialisiert, daß nicht nur eine so weit wie möglich optimale Lösung für den aktuellen Zustand des Datenbanksystems, sondern genauso für zukünftige Anforderungen gefunden werden soll. Die folgenden Betrachtungen beziehen sich dabei auf die Optimierung von Index-Konfigurationen, welche durch ein selbst überwachendes und tunendes Datenbankmanagementsystem durchgeführt werden sollen. Dabei sollen weitgehend die bisherigen statistischen Ansätze der Index- oder Design Advisor genutzt werden, aber in einem dynamischen und autonomen Kontext Verwendung finden. Die daraus neu entstandenen Möglichkeiten sollen zu einer selbständigen Anpassung an sich ändernde Rahmenbedingungen führen. Dies bezieht sich sowohl auf die Daten selbst, auf deren Nutzung sowie auf alle relevanten Aspekte der Systemumgebung.

Dabei muß zwischen dem zu erhaltenden Gewinn dieser Index-Konfiguration und deren Kosten abgewogen werden, das heißt, es wird für jeden Index der Profit bestimmt. Dabei sei eine Menge von Anfragen Q_1, \dots, Q_m und eine Menge von Indexkandidaten I_1, \dots, I_n gegeben. Um den Gewinn eines Indexes I_i für eine Anfrage Q_k zu berechnen, werden die Erzeugungskosten des Indexes I_i für die Anfrage Q_k von den Kosten für die Anfrage Q_k ohne den Index I_i abgezogen. Daraus kann trivialerweise geschlossen werden, daß diese Differenz maximiert werden soll, woraus die Gleichung

$$profit(Q_k, I_i) = \max \{0, cost(Q_k) - cost(Q_k, I_i)\}$$

folgt. Weiterhin sind die Kosten für die Verwaltung eines Indexes I_i zu berücksichtigen, die in den nachfolgenden Betrachtungen als $mcost(I_i)$ bezeichnet werden. Wie leicht nachzuvollziehen ist, sollen nicht nur einzelne Indexkandidaten und deren Gewinn bestimmt werden, sondern auch gesamte Indexkonfigurationen $C \subseteq I_1, \dots, I_j$ von nutzbaren Indexen errechnet werden. Aus diesen Gesichtspunkten ergibt sich die zu maximierende Berechnungsvorschrift

$$\sum_{i=1}^m \max \{profit(Q_i, I_j : I_j \in C)\} - \sum_{I_j \in C} mcost(I_j).$$

Allerdings benötigt die Indexkonfiguration C Speicherplatz im so genannten Index-Pool, wodurch eine Schranke S für den Speicherplatzbedarf für die Indexkonfiguration C eingehalten werden muß. Dies läßt sich wie folgt berechnen

$$\sum_{I_j \in C} size(I_j) \leq S.$$

Problematisch dabei ist, daß es sich um ein NP-Problem [Com78] handelt, und als Variante des Rucksackproblems angesehen werden kann [KPP04]. Aufgrund der Komplexität dieser Probleme gibt es neben exakten Lösungsansätzen (Greedy oder Dynamic Programming) auch approximative Vorschläge [CFM95], die den Aufwand verringern, aber keine optimale Lösung des Problems garantieren können.

3.2 Regelkreis Index-Self-Tuning

Das Self-Tuning wird in aktuellen Ansätzen durch einen Regelkreis [WHMZ94] aus der Überwachung von Systemverhalten und -nutzung (Observation), der Vorhersage zukünftig gewinnbringender Systemeinstellungen (Prediction) und deren Umsetzung (Reaction) realisiert. Speziell für Index-Konfigurationen besteht dieser Regelkreis aus der Überwachung des aktuellen Workloads, der Ableitung geeigneter Indexkandidaten [BC06] und gegebenenfalls der Erzeugung der vielversprechendsten Kandidaten und dem Löschen weniger profitabler Indexe [SSG04].

In **Abbildung 3.1** wird der oben genannte Regelkreis für das Self-Tuning von Bitmap-Index-Konfigurationen beschrieben. Wobei Indexkandidaten durch eine syntaktische Analyse von Anfragen entsprechend der in **Abschnitt 2.2** genannten Anforderungen gesucht werden. Daraufhin werden diese Kandidaten in den Systemtabellen des Datenbankmanagementsystems erzeugt, und für eine What-If-Analyse (was wäre, wenn dieser Index existieren würde) dem Optimierer zur Verfügung gestellt. Aus der What-If-Analyse berechnet sich der Gewinn der Indexkandidaten und der bereits materialisierten Indexe gegenüber der Ausführung selbiger Anfrage ohne Indexe, und dieser wird für spätere Analysen gespeichert (Observation).

Der gewonne Nutzen muß auf der Basis eines Kostenmodells analysiert und bewertet werden, um die Entscheidung treffen zu können, ob eine neue Index-Konfiguration verwendet wird oder nicht. Dahingehend soll ein Kostenmodell eine Entscheidung unterstützen, ob es wirklich rentabel ist eine alte Index-Konfiguration durch eine Neue zu ersetzen (Prediction). Denn ein zu geringer Gewinn in der Anfrageverarbeitung würde insgesamt durch

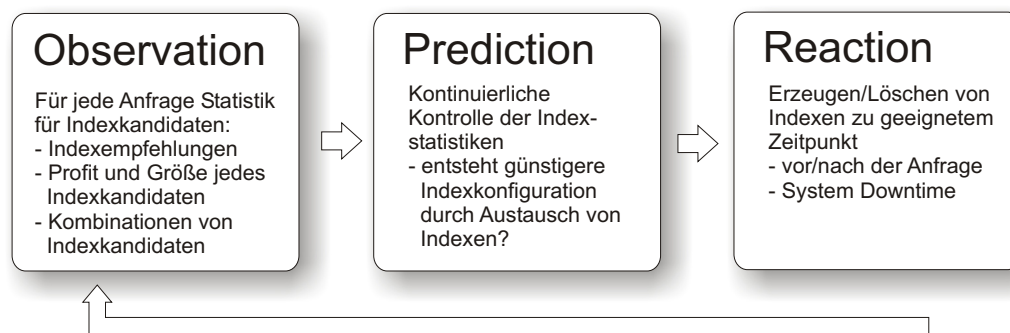


Abbildung 3.1: Regelkreis Index-Self-Tuning

die Kosten für die Abschätzungen und Erstellung zu einem unrentablen Prozeß und sehr häufiger Veränderung der Indexe führen.

Zum Abschluß der Verarbeitung der Index-Empfehlungen muß eine Analyse des Anwendungsfalles vollzogen werden, wobei in die Betrachtungen der Aufwand für das Erstellen und Löschen von Indexen einbezogen wird. Als Resultat dieser Untersuchungen muß die Entscheidung getroffen werden, ob und wann geeignete Indexe erstellt werden.

Innerhalb dieses Ablaufes ist es genau das Zusammenspiel verschiedener Parameter, die in einem Kostenmodell zusammenfließen, um einen realen Gewinn über den Gesamtprozeß bilden zu können. Der Ablauf dieses Prozesses wird durch einen vorher definierten Zeitrahmen oder vorgegebene Ereignisse fortlaufend wiederholt.

3.3 MAPE von IBM

Im Laufe dieses Abschnittes wird der Ansatz **MAPE** von IBM erläutert. Dabei steht **MAPE** für Monitor Analyze Plan Execute, und es werden von IBM sehr ähnliche Ziele verfolgt, wie sie mit Hilfe des Regelkreises (siehe **Abschnitt 3.2**) umgesetzt wurden.

In **Abbildung 3.2** wird die Architektur des von IBM vorgestellten Ansatzes zum Online-Tuning [Cor05] dargestellt. Es ist zu erkennen, daß im Gegensatz zum Regelkreis vier Etappen im Optimierungsprozeß durchlaufen werden müssen. Die Aufgabe der ersten Stufe, dem Monitoring, entspricht der entsprechenden Stufe des Regelkreises. Dabei werden Statistiken über die Daten und das Datenbanksystem selbst gesammelt und zum Beispiel gefiltert. Dieser Prozeß läuft in einem kontinuierlichen Zeitrahmen ab, bis ein Objekt identifiziert wird, das Auffälligkeiten aufweist, so daß es analysiert werden muß. Die nächste Stufe heißt Analyse und tritt in Kraft, wenn der Monitor-Prozeß Alarm geschlagen hat. Die Werkzeuge für die Analyse von Auffälligkeiten befinden sich in dieser Stufe des Gesamtprozesses. Weiterhin wird in diesem Teil des Optimierungsprozesses festgestellt, ob eine Veränderung durchgeführt werden muß. Ist dies der Fall, dann werden die Information an die nächste Stufe übergeben, und die Verarbeitung der festgestellten Ungereimtheit wird weitergereicht.

Wie in **Abbildung 3.2** zu sehen ist, wird diese Operation durch eine so genannte Change Request durchgeführt. Erhält der Planner (Stufe Plan) dieses Request, so erzeugt oder wählt er für das auffällige Objekt (Reihe von) Operationen aus. Diese werden an dieser Stelle organisiert und für spätere Verarbeitungen vorbereitet, zum Beispiel mit Hilfe

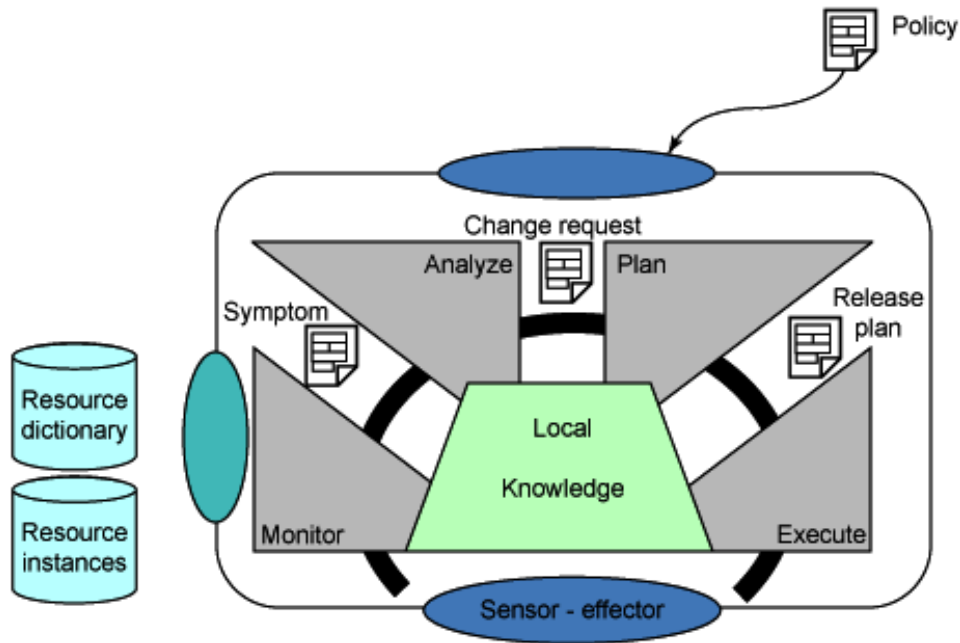


Abbildung 3.2: Darstellung des Zyklus des MAPE-Ansatzes

von Freigaben oder Sperren bis hin zu vollständigen Ausführungsplänen. Der Aufbau des erzeugten Planes kann stark variieren, und hängt vom Objekt und dessen Veränderung ab, denn es kann sich um einzelne Befehle oder aber um komplizierte Abläufe handeln. Das Ergebnis, der so genannte Release Plan, enthält alle nötigen Informationen für die Ausführung, welche an die nächste Stufe übergeben werden.

Die letzte Stufe dieses Ansatzes wurde mit Execute bezeichnet. Dabei sind in dieser Stufe alle Mechanismen für das ausführen und organisieren dieses Plans enthalten. Nach Erhalt des Release Plans übernimmt ein autonomer Manager die Verwaltung der durchzuführenden Änderungen. Dies ist notwendig, denn es ist möglich, daß verändernde Maßnahmen ergriffen werden müssen, um ein oder mehrere Objekte ändern zu können. Der autonome Manager wandelt dabei den Release Plan in eine Serie von Ausführungen um, die vom Datenbankmanagementsystem verstanden und ausgeführt werden können. Des Weiteren kann es notwendig sein die lokale Wissensbasis (Local Knowledge) zu erweitern oder zu aktualisieren.

Der Unterschied zum zuvor vorgestellten Regelkreis ist die systemnahe Aufteilung der Aufgaben innerhalb des Optimierungsprozesses, wodurch bei MAPE ein Arbeitsschritt in zwei Phasen geteilt wurde.

3.4 Physical Design Alerter von Microsoft

Dieser Abschnitt wird den von Surajit Chaudhuri und Nicolas Bruno (Microsoft Research) vorgestellten Ansatz eines Physical Design Alerters von Microsoft erläutern [BC06]. Das verfolgte Ziel dabei ist, den Datenbankadministrator (**DBA**) in seiner Arbeit zu entlassen, um so den Personalaufwand zu verringern. Dabei soll erreicht werden, daß der Physical Design Alerter suboptimale Konfigurationen eines Datenbanksystems erkennt. Der praktikable Ansatz daran liegt in der Veränderung der Anforderungen an ein Da-

tenbanksystem, welche durch den Alerter erkannt werden sollen. Dadurch soll es möglich sein die Ausführung teurer Tuning-Tools zu vermeiden, und eine neue optimale Konfiguration durch den Physical Design Alerter zu finden.

Dabei ist die Alarmierung durch dieses Tool in verschieden komplexe Stufen eingeteilt. Die erste Stufe setzt die bisherige Arbeit eines Datenbankadministrators in automatisierter Form um, denn der Alerter wird periodisch vom Datenbankadministrator aufgerufen, wenn dieser der Meinung ist eine Veränderung in der Konfiguration könnte notwendig geworden sein (Low Overhead). Bei der zweite Variante handelt es sich um eine einfach Alarmfunktion, die nur die Aufgabe hat zu signalisieren, daß ein bessere Konfiguration möglich ist. Dazu wird davon ausgegangen, daß eine Optimierung durch ein Tuning-Tool mindestens ebenso groß ist wie die die des Alerters, denn dieser muß erst Suboptimalität im laufenden Betrieb feststellen (Lower Bound). Die mögliche Stufe des Physical Design Alerters erlaubt in Abhängigkeit von den Kosten, die während der Anfrageoptimierung toleriert werden, verschiedene Stufen der Annäherung an die obere Grenze (Upper Bound) der optimal-erreichbaren Werte (Konfiguration).

In **Abbildung 3.3** wird gezeigt wie sich der Physical Design Alerter in den Monitor-Diagnose-Tune-Zyklus einfügt. Die Anfragen werden weiterhin direkt an das Datenbankmanagementsystem gestellt, welches die Ergebnisse ohne Eingreifen des Alerters zurückgibt. Das Datenbankmanagementsystem sammelt mit Hilfe von Statistikfunktionen die

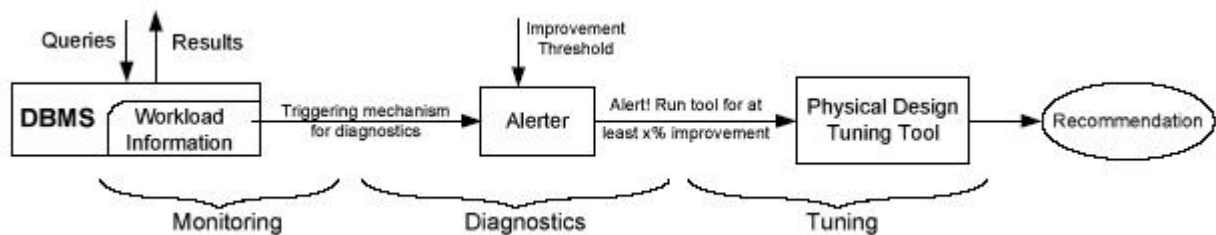


Abbildung 3.3: Monitor-Diagnose-Tune-Zyklus des Physical Design Problem [BC06]

Workload Informationen, wodurch es alleinig für das Monitoring verantwortlich ist. Die gewonnenen Informationen werden mittels Auslösermechanismus (Trigger) für die spätere Diagnose bereitgestellt.

An dieser Stelle beginnt die Aufgabe des Physical Design Alerters, denn er erhält die Informationen vom Datenbankmanagementsystem, und führt anhand der Voreinstellungen und Vorgabe die Analysen durch. Tritt der Fall ein, daß der Alerter eine nicht optimale Konfiguration feststellt, dann schlägt dieser Alarm. Diese Information und die Angabe der möglichen Verbesserung (in Prozent) wird an ein Tuning Tool weitergegeben.

Dieses Tuning Tool (Physical Design Tuning Tool) erhält sämtliche Informationen des Alertes und analysiert diese. Daraufhin führt dieses Tool eine Optimierung der Konfigurationen durch, und gibt als Ergebnis den Vorschlag für eine neue verbesserte Konfiguration aus.

Der Ansatz des Physical Design Alerters wurde von seinen Entwickler derart erweitert, daß die physikalische Optimierung des Datenbanksystems Online stattfinden kann. Dies bedeutet also, daß basierend auf den Informationen des Alerters ein Tuning Tool erweitert werden soll [BC07].

Für die Optimierung werden zur Ausführung von Anfragen so genannte Zugriffs- oder Anfragepläne erzeugt. Diese werden in einzelne Operation beziehungsweise Teilbäume

aufgeteilt, wodurch die nötigen Informationen für eine Optimierung von Indizes gewonnen werden können. Anschließend müssen anhand von Kostenmodellen und Optimierungsstrategien vielversprechende Empfehlungen gefunden werden.

Dabei wird zunächst zwischen einzelnen Index-Szenarien und Szenarien mit Index-Interaktion unterschieden. Nach eingehender Analyse kann anhand eines Kostenmodells für die einzelnen Index-Szenarien schnell eine Entscheidung für eine Index-Empfehlung ausgegeben werden. Bei Index-Interaktionen dagegen ist es noch notwendig die Überschneidungen der Indizes zu analysieren. Dabei wird zum Beispiel geprüft, ob einer der Indizes Teil eines anderen Indexes ist. Nach Beendigung der Analyse der Interaktion werden gewonnenen Werte aus der Kosten-/Nutzen-Analyse nachträglich justiert.

Kapitel 4

Konzept für das Self-Tuning von Bitmap-Indexen

In diesem Kapitel sollen die Vorschläge für neue Konzepte dargelegt und erläutert werden. Dabei gibt der Bezug auf die Grundgedanken, die zur Analyse und Weiterführung der vorhandenen Problemstellung führten, einen Einblick in die folgende Thematik und weiteren Ausführungen. Daraufhin folgen Lösungsvorschläge für die verschiedenen Teilprobleme, das Gesamtkonzept als Ergebnis dieser Betrachtungen und die Veränderung in der Analyse des Workloads. Anschließend werden die Unterschiede zu bisherigen Vorschlägen diskutiert, und die Anforderungen und Grenzen des neuen Modells betrachtet und ausgewertet.

4.1 Herangehensweise und Entwurf eines Kostenmodells

Das Index-Tuning als Teilbereich der Datenbankoptimierung ist eine der wichtigsten und meist betrachteten Problemstellungen innerhalb der Self-Tuning-Forschung, da die Resultate sich maßgeblich auf die Antwort- und Anfragebearbeitungszeiten auswirken. Oft wird dieses Problem auch als Index Selection Problem (**ISP**) bezeichnet, wobei es sich wie in dieser Arbeit um die Bestimmung von Index-Konfigurationen handelt (siehe **Abschnitt 3.1**). Diese sollen möglichst eine Menge von Anfragen, auch Workload genannt, eines gegebenen Anwendungsfalles oder Zustandes des Datenbanksystems bestmöglich unterstützen. Dieses Problem ist schon seit Jahren bekannt und wird in der Literatur behandelt, aber bisher wurde es als statisch angesehen. Es wurde davon ausgegangen diesen Entwurf einmalig durchzuführen, und diese Konfigurationen stetig (statisch) zu verwenden. Dies führte zu der Entwicklung so genannter Index- und Design Advisor, die einen gegebenen Workload untersuchen und daraufhin eine Index-Empfehlung ausgeben. Aber die Frage zur Erstellung oder Nichterstellung dieser Indexe blieb nach wie vor in der Entscheidungsgewalt eines Datenbankadministrators.

Aufgrund der stetigen Weiterentwicklung von Datenbanksystemen und der sich verändernden Umgebung und Anforderung an sie, ist diese Herangehensweise oft nicht optimal oder gar völlig falsch für den aktuellen Zustand des Systems. Diese Tatsache

läßt sich trivial nachvollziehen, wenn der Bezug zu heute weit verbreiteten Systemen des OLAP-Ansatzes hergestellt wird. Denn hier ist die Menge von Anfragen schwer abzuschätzen, da sich diese ständig an den gewünschten Analysezweck anpassen. Es ist leicht ersichtlich, daß sich zum Beispiel in der heutigen Wirtschaft die Anforderungen an die verwendeten OLAP-System häufig verändern [CCS93]. Des Weiteren sind neue Szenarien immer weiter verbreitet, in denen die Abschätzung zukünftiger Anfragen sehr schwierig ist, da diese sich explorativ gestalten. Das heißt es finden viele Ad-Hoc⁷ Anfragen statt, die zu einem nicht vorhersehbaren Zeitpunkt und Anfrageform auftreten können. Dazu werden selbst relativ statische Szenarien durch steigende Anforderungen und Schnellebigkeit immer dynamischer. Weiterhin wird die Performance von Index-Strukturen durch Veränderungen am Datenbankschema oder anderen Tuningmaßnahmen (etc.) gegebenenfalls negativ beeinflusst. Dies macht die Anwendung von bisherigen Ansätzen und Entwicklungen, wie einem Index- oder Design Advisor, immer schwieriger und personalaufwendiger. All diese Ausführungen haben also zur Folge, daß das Index Selection Problem mehrfach zu nicht vorhersehbaren Zeitpunkten gelöst werden muß. Dies bedingt eine ständige Überwachung des aktuellen Workloads (Observation) und falls nötig eine darauf folgende neue Index-Empfehlung (Prediction). Im idealsten Fall würde das Datenbankmanagementsystem diese dann autonom verarbeiten, und nicht auf eine Entscheidung des Datenbankadministrators in Bezug auf die Erstellung der empfohlenen Index-Konfiguration (oder eben nicht) warten müssen (Reaction) [LSS07] (siehe auch **Abschnitt 3.1**).

Durch die wiederholte (dynamische) Lösung des Index Selection Problems ist der hohe Aufwand für diese Berechnungen zu beachten, wodurch sich die Frage stellt, zu welchem Zeitpunkt Entscheidungen getroffen werden sollen. Als Erstes ist dabei zu berücksichtigen wann und wie der Workload überwacht wird (Observation). Findet die Überwachung manuell und statisch statt, und wird diese eventuell wiederholt, das heißt es werden Techniken des Index Advisors genutzt. Oder aber wird ein Monitorprozeß nach Ablauf eines gewissen Zeitintervalls beziehungsweise durch ein bestimmtes Ereignis (diskret) angestoßen, dann ist die Überwachung unabhängig von menschlichen Eingriffen. Die letzte und aufwendigste Möglichkeit ist die dauerhafte und somit kontinuierliche Überwachung der Anfragen, aber so kann erreicht werden, daß die Beobachtungen, und damit die Grundlage für spätere Entscheidungen, am genauesten sind. Diese Überlegungen lassen sich ohne Weiteres auf die Entscheidungsphase (Prediction) übertragen. Es kann sich um eine einmalige Entscheidung handeln, die unter gewissen Umständen erneut durchgeführt wird. Andererseits kann diese Phase als Prozeß angesehen werden, der durch Ereignisse (Anfragen oder bestimmte Operationen) gesteuert wird. Für die dritte Phase stellt sich die Frage, zu welchem Zeitpunkt oder in welchem Zeitintervall die getroffenen Entscheidungen umgesetzt werden sollen, wobei dies durch den Zustand des Systems bestimmt sein kann. Dabei muß festgelegt werden, ob die neuen Index-Empfehlungen vor oder nach dem auslösenden Ereignis umgesetzt werden, was wiederum Einfluß auf die Berechnung der Kosten und des Nutzens hat. Denn entweder verzögert sich die Anfrage durch die Erstellung von Indexen oder der Nutzen der Indexe tritt für die aktuelle Ausführung nicht ein. Wenn es keine generelle Entscheidung des Erstellungszeitpunktes gibt, dann könnte dies durch eine Abschätzung des Gewinnes gegenüber den Erstellungskosten entschieden werden. Weiterhin ist zu beachten, ob die durchzuführende Operation oder Anfrage als

⁷Spontan, subjektiv, nicht vorhersagbar

zeitkritisch angesehen werden muß. Eine weitere Möglichkeit ist es die Anfrage selbst für die Erstellung der Indexe zu nutzen, da auf die Datenbankobjekte schon während der Anfrage zugegriffen wird, allerdings würde dies die Anfragebearbeitungszeit⁸ gegebenenfalls negativ beeinflussen. Weniger autonom dagegen stellt sich die letzte Möglichkeit dar, denn diese findet zeitverzögert statt. Die Erstellung wird dabei entweder manuell angestoßen oder zu Zeiten geringer Last beziehungsweise so genannten Offline-Zeiten (wenn diese Phasen für das System existieren) [LSSS07]. Die Gesamtheit dieser von der Zeit bestimmten Einteilung hängt dabei vom jeweiligen Anwendungsszenario, dessen Strategien, die verfolgt werden sollen, und den betrachteten Objekten des Datenbanksystems ab.

Für die Umsetzung der bisherigen Betrachtungen sollen bestehende Systeme und Ansätze genutzt werden, um so eine Skalierbarkeit der Problemstellung gewährleisten zu können. Weiterhin kann so erreicht werden, daß für die Optimierung des Datenbanksystems selbst, und die durch die, in dieser Arbeit vorgestellten, Ansätze vollzogene Optimierung, die gleichen Kostenmodelle und Analysedaten verwendet werden. Dies soll eine spätere Einbindung von Lösungsvorschlägen in bestehende Datenbankmanagementsysteme fördern, da eine solche Integration bisher nicht stattgefunden hat, und die entstandenen Systeme autonom von Systemen wie dem internen Optimierer oder Index Advisor arbeiteten.

4.2 Profit von Index-Empfehlungen

In diesem Abschnitt sollen weitere Probleme im Bereich des Index-Tunings diskutiert werden, um so einen objektiven Blick auf weitere Vorschläge und Analysen zu gewährleisten.

Zuerst läßt sich die Problematik erkennen, wie mit Indexen umgegangen werden muß, die sich gegenseitig überschneiden. Denn es ist nicht unüblich, daß auf einer Relation R ein Index über dem Attribut $R(A)$ existiert, und zusätzlich ein weiterer Index über $R(A, B)$ definiert ist. Wie leicht zu erkennen ist, überschneiden sich beide Indexe im Bereich $R(A)$, und somit kann der Profit des Indexes über $R(A)$ ebenso dem Index über $R(A, B)$ angerechnet werden. Dazu ist allerdings die formale Feststellung nötig, ob ein Index I_m in Index I_n enthalten ist. Dies ist genau dann der Fall, wenn beide Indexe über dieselbe Relation R definiert sind, und die Attribute von I_m einen Präfix, das heißt ein Bestandteil vor den ungleichen Teilen, von I_n bilden. Zusätzlich ist an dieser Stelle die Sortierung der Indexe zu beachten, da diese aufsteigend oder absteigend geordnet sein können, und sie in beiden Fällen gleich sein muß.

Weiterhin muß an dieser Stelle geklärt werden, wie die neuen Index-Empfehlungen nicht nur gegenüber einem System ohne bestehende Indexe, wie es beim Entwurf der Fall ist, oder bisher nicht indextierten Attributen bewertet werden, sondern diese ebenfalls gegen bestehende Optimierungen des Datenbankmanagementsystems gewichtet werden können. Dazu wird eine Anfrage Q zunächst ohne die neuen Index-Empfehlungen mit den konventionellen Methoden optimiert und deren Kosten $cost(Q)$ bestimmt. Danach kann diese Anfrage analysiert werden, um so eventuell speziell auf die bisherigen Indexe reagieren zu können, und die aktuelle Konfiguration gegebenenfalls nur anzupassen statt eine komplett Neue erstellen zu müssen. Anschließend wird die Anfrage Q erneut mit den neuen Indexkandidaten I_1, \dots, I_k analysiert und optimiert, wodurch eine Abschätzung der

⁸Abhängig von Betrachtungen gegenüber anderen Möglichkeiten

Kosten $cost(Q, I)$ bestimmt werden kann. Dabei handelt es sich nur um eine Schätzung, da die Materialisierung der Indexe bisher noch nicht stattgefunden hat, und die Optimierung zu diesem Zeitpunkt nur virtuell durchgespielt wird. Die Kosten für die Index-Empfehlung $E = I_1, \dots, I_k$ ergeben sich dabei aus den einzelnen Kosten der Indexkandidaten, und führen damit zur einer in **3.1** angelehnten Berechnung für den Profit der Indexkonfiguration $profit(Q, E)$. Dieser ergibt sich dann aus der Differenz der Kosten für die Anfrage $cost(Q)$ und den Kosten für die Index-Empfehlung $cost(Q, E)$, daraus folgt dann

$$profit(Q, E) = cost(Q) - cost(Q, E).$$

Als Nächstes soll diskutiert werden, wie unter den einzelnen Indexkandidaten I_1, \dots, I_k der Gesamtprofit $profit(Q, E)$ aufgeteilt werden kann, denn dies blieb bisher offen. Es wurden dazu schon verschiedene Alternativen untersucht [SSG04], dabei stellte sich die Größe eines Indexes $size(I)$ als gut geeignetes Aufteilungsmaß heraus. Dies bedeutet, daß der Profit des Indexes $I \in E$ aus dem Gesamtprofit $profit(Q, E)$ mit seiner Größe gegenüber der Summe der Größe der Indexe aus E gewichtet wird. Wie sich nachvollziehen läßt ergibt sich daraus die Formel

$$profit(Q, I) = \frac{profit(Q, E) * size(I)}{\sum_{I_j \in E} size(I_j)}$$

für die Berechnung. Dazu ist zu sagen, daß es sich dabei nur um eine Approximation des Profits handelt. Denn es ist leicht zu erkennen, daß der gesamte Profit der Index-Empfehlung E im Allgemeinen ungleich der Aufsummierung der einzelnen Profite der Indexe I ist. Folglich ergibt sich daraus

$$profit(Q, E) \neq \sum_{I_j \in E} profit(Q, I).$$

Denn zum Beispiel bei der Verwendung eines Joins, der die Sortierung zweier Indexe über die Join-Attribute dieses Joins ausnutzt, ist der Profit im Allgemeinen höher, als wenn jeweils nur einer dieser beiden Indexe vorhanden wäre, und aus den einzelnen Profiten die Summe gebildet werden würde.

Einen weiteren problematischen Teil bilden die so genannten Update-Operationen, diese werden in [LSS07] für Baumverfahren vorgestellt. Für Bitmap-Index-Strukturen bedarf es allerdings genauerer Betrachtungen, aber an dieser Stelle soll auf diesen Teilaspekt vorerst nicht eingegangen werden, da dies im späteren Verlauf der Arbeit im Speziellen der Fall sein wird.

Da die Lösung des Index Selection Problems, wie zuvor besprochen, sehr aufwendig ist, kann es hilfreich sein, die Analyse nicht nach jeder Anfrage (vorausgesetzt sie ist aufwendig genug) durchzuführen. An Stelle dessen kann die Lösung dieses Problems in verschiedenen ereignisgesteuerten Zeitabständen (siehe auch **4.1**) geschehen. Auf diese Art und Weise könnte die Komplexität dieses Problems reduziert werden, allerdings wäre es auch weiter denkbar, daß die Analyse über eine repräsentative Stichprobe von Anfragen ausgeführt wird. Aber es bleibt zu diskutieren, was in diesem Bereich als repräsentativ genug zu bezeichnen ist, da gerade autonome und dynamische Optimierung auch für kurzfristige Änderungen das Ziel darstellt. Dennoch bleibt die entscheidende Frage, wie mit der Situation der Komplexität der Optimierung umgegangen werden soll, da selbst bei immer weiter steigender Hardwareperformance die Anforderung an diese

ebenbürtig steigen.

Der letzte Aspekt dieses Abschnittes befaßt sich mit der Problematik des so genannten Agings [NS01], welches auch aus dem Bereich der Scheduler für Transaktionen von Datenbankmanagementsystemen oder Betriebssysteme bekannt ist. Dabei soll dafür gesorgt werden, daß ältere Index-Empfehlungen mit der Zeit an Gewichtung in deren Bewertung verlieren, da diese gegebenenfalls nicht zum aktuellen Zustand der Datenbank passen. Dadurch kann auf kurzfristige Veränderung wesentlich besser oder kurzfristiger reagiert werden, denn jüngere Index-Empfehlungen können durch ihre bessere Gewichtung frühzeitiger umgesetzt werden. Auch dies ist ein bekanntes Problem aus den oben beschriebenen Bereichen, und in [LSS07] wurde vorgeschlagen dies in Epochen zu lösen. Dabei wird mit Hilfe von verschiedenen Zeitstempeln ein Gewinn für einzelne Anfragen aus dem zuvor bekannten Profit für deren Indexe berechnet.

4.3 Anfrageanalyse für Bitmap-Index-Empfehlungen

Für die, in den vorherigen Abschnitten beschriebenen, Empfehlungen ist es (wie bei jeder Art der Anfrageoptimierung) nötig die Anfrage selbst oder eine Menge von Anfragen zu analysieren. Daher werden sich die folgenden Betrachtungen dieses Abschnittes, aufbauend auf den bisherigen Feststellungen und Analysen, dieser Aufgabe stellen. Wie zuvor beschrieben, werden dabei bisherige Ansätze und Möglichkeiten der Datenbankmanagementsysteme genutzt. Dabei soll auf die speziellen Betrachtungen und Anforderungen für Bitmap-Index-Strukturen und deren Konfigurationen eingegangen werden.

Zunächst werden die nötigen Betrachtungen für die Such- und Zugriffsindexe auf Bitmap-Basis beschrieben. Dazu ist als Erstes zu sagen, daß die Bedingungen für die Erstellung eines Indexes, wie bei jeder anderen Index-Struktur auch, erfüllt sein müssen. Dazu gehört vor allem, daß das zu indexierende Attribut ein für das Datenbankmanagementsystem sortierbares Format (zum Beispiel Zahlen oder eine alphabetische Reihenfolge) besitzt. Weiterhin sollte das Attribut mehr als einen Wert enthalten, denn wie leicht nachzuvollziehen ist, wird der Erfolg eines Indexes über einen Wert des Attributes mit einer beliebigen Anzahl von Einträgen ausbleiben. Zusätzlich darf ein Wert dem Attributs nicht mehrmalig in unterschiedlichen Zusammenhängen auftreten, oder er ist selbst Teil einer Schlüsselbedingung, das heißt es wird der so genannte UNIQUE-Eigenschaft (Einzigartigkeit) erfüllt (siehe **Abschnitt 2.1**), denn ansonsten wäre eine eindeutige Zuordnung dieses Wertes im Index selbst nicht mehr möglich. Des Weiteren gilt speziell für Bitmap-Index-Strukturen eine weitere Bedingung, denn wie in **Abschnitt 2.2** gezeigt, ist es für Bitmap-Strukturen ungeeignet viele unterschiedliche Werte des Attributes zu indexieren. Dies läßt sich aus der Eigenschaft schließen, daß für jeden Wert ein Bitvektor angelegt wird, und eine sehr hohe Menge von Bitvektoren schwierig zu verwalten ist. Allerdings ist diese Aussage nicht im Allgemeinen gültig, da es nachzuvollziehen ist, daß bei wachsender Anzahl von Tupeln auch die Menge möglicher Attributwerte steigen kann. Aufgrund dessen ist es nötig diese Aussage zu formalisieren, und diese wird im Folgenden als Attributwertkardinalitätsbedingung bezeichnet. Diese Bedingung wird nun genau aus dem Verhältnis der Kardinalität der Domäne (Wertebereich) des Attributes $card(dom(A))$ und der Kardinalität (oder Größe) der Relation $card(r(R))$ bestimmt,

und muß den Schwellwert der maximalen Selektivität⁹ $maxSelectivity$ unterschreiten. Dadurch ist es möglich eine Abschätzung mit Hilfe der Formel

$$card(dom(A))/card(r(R)) < maxSelectivity$$

für die Eignung von Bitmap-Indizes durchzuführen. Für dieses Sachverhalt haben verschiedene Fallstudien [DB06] gezeigt, daß die Kardinalität des Wertebereiches eines Attributes circa einen Prozent der Anzahl der Tupel der Relation betragen darf.

Anschließend soll nun die Anfrageanalyse für die so genannten Join-Indizes (siehe **Abschnitt 2.5**), die zweite Art von Indizes in dieser Arbeit, behandelt werden. Dabei soll auf die speziellen Anforderungen und Analyseaufgaben eingegangen werden, allerdings soll dies auch an dieser Stelle nur speziell für die Bitmap-Index-Strukturen geschehen. Auch an diesem Punkt ist es nachvollziehbar, daß die Join-Bedingungen der beiden (oder Menge von) zu verbindenden Attribute(n) wie bei jedem anderen Join-Index ebenso erfüllt werden müssen. Das bedeutet, daß es sich bei beiden Verbundpartnern um Attribute handeln muß, die denselben Sachverhalt abbilden. Dazu gehört auch, daß eine Vergleichsoperation, die von der Art des Joins abhängt, zwischen den Tupeln der beiden Attribute möglich ist, was durch unterschiedliche Datentypen oder Inhalte verhindert werden könnte. Zur Veranschaulichung seien Relation $R_1(A, B)$ und Relation $R_2(B, C)$ gegeben, wobei es sich bei Attribut B um das Verbundattribut der Relationen R_1 und R_2 handelt, und die obigen Bedingungen erfüllt seien. Dann würde folglich die Relation $R_{12}(A, B, (B), C)$ entstehen, in der je nach Verbundtyp nur der Join-Bedingung entsprechende Tupel aus beiden Relationen enthalten sind. Falls beide Attribute unterschiedliche Namen tragen, dann muß das Datenbankmanagementsystem gegebenenfalls über META-Informationen die Feststellung treffen, daß es sich um mögliche Join-Partner handelt. Vorausgesetzt die Join-Bedingungen sind für die Anfrage oder Menge von Anfragen erfüllt, dann kann ein Join im Allgemeinen stattfinden. Allerdings treffen, wie in **Abschnitt 2.2** beschrieben wurde, für Bitmap-Join-Indizes wie auch für Such- oder Zugriffsindizes auf Bitmap-Basis nur Bedingungen der so genannten WHERE-Klausel zu. Das heißt es muß ein so genannter (innerer) Equi-Join $r(R) \bowtie_{A_1=A_2} r(S)$ erkannt werden, denn mit Hilfe von Bitmap-Indizes ist wie in **Abschnitt 2.2** beschrieben nur diese Vergleichsoperation möglich. Weiterhin muß erfüllt sein, daß Attribut A_1 und Attribut A_2 wie oben beschrieben als Verbundattribute erkannt werden, sowie $A_1 \in R$ und $A_2 \in S$ gilt. Es ist allerdings denkbar, daß weitere Join-Methoden mit Hilfe von Bitmap-Index-Varianten durchführbar sind.

Wie für jeden Bitmap-Index muß auch für die Verbundattribute die zuvor beschriebene Attributwertkardinalitätsbedingung erfüllt werden. Das heißt wenn beide einzeln diese Bedingung erfüllen, dann kann von einer guten Performance für den Bitmap-Join-Index und ebenso von der Erfüllung für das Join-Ergebnis ausgegangen werden. Aber auch bei dieser Aussage kann nicht von einer allgemeinen Gültigkeit ausgegangen werden, denn es gibt an dieser Stelle unterschiedliche Fälle zu betrachten.

*Fall 1: $A_1 \in R \wedge A_2 \in S$ erfüllen **AWK**-Bedingung*

Der erste Fall tritt ein, wenn, wie oben beschrieben, beide Verbundpartner die Attributwertkardinalitätsbedingung erfüllen. Dann bedarf es für die Anfrageanalyse nur noch einer Prüfung anhand von Kostenmodellen, wie sie in vorherigen Abschnitten beschrieben sind.

⁹Maximale Prozentsatz von Tupeln als Anfrageergebnis gegenüber der Größe der Relation

*Fall 2: $A_1 \in R \wedge A_2 \in S$ erfüllen **AWK**-Bedingung nicht*

Der zweite Fall wäre gegeben, wenn keiner der Join-Partner diese Bedingung für Bitmap-Index-Strukturen erfüllt. Daraus läßt sich schließen, daß es sehr unwahrscheinlich ist, daß das Verbundergebnis diese erfüllt, sowie dessen Berechnung mit Hilfe von Bitmap-Indexen eine performante Lösung darstellt. Dies kann durch ein Zugreifen auf Statistiken und Abschätzungen geprüft werden, soll aber aufgrund der geringen Wahrscheinlichkeit dieses Falles nicht weiter in die Betrachtungen einfließen.

*Fall 3: $A_1 \in R \vee A_2 \in S$ erfüllen **AWK**-Bedingung*

Wenn nun einer der Verbundpartner die Attributwertkardinalitätsbedingung erfüllt, dann tritt ein dritter denkbarer Anwendungsfall ein. An dieser Stelle stellt sich dann die Frage, welche Bedeutung einerseits die Erfüllung des einen Verbundpartners und andererseits die Nicht-Erfüllung des anderen Verbundpartners hat. Als gute Grundlage für eine solche Entscheidung läßt sich die Kardinalität der Join-Partner heranziehen, dazu werden zum Beispiel die Kardinalitäten der Relationen R und S gebildet und ins Verhältnis gesetzt. Nun ist die Frage, wie diese Betrachtungsweise zu einer Entscheidung beitragen kann. Wenn nun einer dieser Verbundpartner wesentlich mehr Tupel besitzt als der Andere, dann bedeutet dies, daß der größere der Join-Partner eine wichtigere Rolle für die Berechnung der Kosten des Verbundes übernimmt. Diese Tatsache ist trivial nachzuvollziehen, da mehr Tupel für die Verbundoperation mehr Kosten verursachen, und diese zu größeren Teilen einem der Verbundpartner zugeordnet werden können. Daraus läßt sich also der Schluß ziehen, daß es für diesen Anwendungsfall ausreichend sein kann, wenn die Attributwertkardinalitätsbedingung für den größeren (Anzahl der Tupel) der Verbundpartner erfüllt ist. Denn dieser ist dann bestimmend für die Kosten des gesamten Joins und das Join-Ergebnis, das dann durch den Einfluß des größeren Verbundpartners mit großer Wahrscheinlichkeit ebenfalls diese Bedingung erfüllt. Die Wahrscheinlichkeit dafür sowie die Performance des Joins hängen dabei vom Verhältnis der Kardinalitäten ($card(r(R))$ und $card(r(S))$) ab. Eine enorm unterschiedliche Kardinalität von Relationen ist kein selten eintretender Fall, als Beispiel soll das so genannte STAR-Schema (siehe **Abschnitt 2.5**) herangezogen werden. Es sei die Relation F als Faktentabelle und eine Dimensionstabelle D des Star-Schemas gegeben, dabei beträgt die Kardinalität der Faktentabelle ($card(r(F)) = 10.000.000$ (also 10.000.00 Tupel) und für die Dimensionstabelle $card(r(D)) = 100$ (100 Tupel). Setzt man diese wie oben beschrieben ins Verhältnis zu einander, dann erhält man

$$card(r(F))/card(r(D)) = 10000.$$

Dies bedeutet also, daß die Relation F um den Faktor 10000 mehr Tupel besitzt als die Relation D , wodurch bei einer Abschätzung der Kosten des Verbundes der Einfluß durch die Relation F um den Faktor 10000 höher ist als das Gewicht der Relation D . Daraus kann gefolgert werden, daß zu Gunsten der größeren Relation F und der Berechnung des Joins $r(F) \bowtie r(D)$ die ineffektive Indexierung der Relation D in Kauf genommen werden kann, da diese für die Gesamtperformance ein zu geringes Gewicht besitzt. Diese Feststellung zieht wiederum genauere Betrachtungen für die Anfrageanalyse nach sich, denn es muß eine Abschätzung über den Einfluß einer Relation für das Join-Ergebnis getroffen werden. Allerdings kann im Allgemeinen gesagt werden, daß je größer das Verhältnis zu Gunsten der Relation, die die Attributwertkardinalitätsbedingung erfüllt, ausfällt, desto

mehr ist das Gewicht des kleineren Verbundpartners für die Join-Berechnung zu vernachlässigen. Diese Betrachtungen lassen weitere Attribute der Relationen außer Acht, und beziehen sich nur auf die Verbundattribute selbst.

*Fall 4: $r(R) \bowtie_{A_1=A_2} r(S) \wedge A_3 \in S$ erfüllt **AWK**-Bedingung*

Die vierte Möglichkeit ist ein Vorschlag von Oracle [LS02] in dem die bisherigen Betrachtungen nicht zutreffen, denn hier wird ein drittes Attribut $A_3 \in S$ (siehe obiges Beispiel $r(R) \bowtie_{A_1=A_2} r(S)$), welches ein Bestandteil der WHERE-Klausel sein muß, in die Betrachtungen mit einbezogen. In diesem Ansatz muß das Attribut A_3 die Attributwertkardinalitätsbedingung erfüllen, da mit Hilfe dieses Attributes (Indexschlüssel) der von Oracle vorgeschlagene Bitmap-Join-Index erstellt wird. Allerdings müssen die Attribute A_1 und A_2 nun nicht mehr Bestandteil einer WHERE-Klausel sein, denn dies wird durch den Bitmap-Join-Index realisiert. Dieser Ansatz zielt speziell auf Anwendungen im Bereich Data-Warehouses (siehe **Abschnitt 2.5**) ab, weshalb für die folgenden Erläuterungen ein STAR-Schema gewählt wurde. Dabei seien Relation R als Faktentabelle und eine Dimensionstabelle Relation S gegeben. Wurden also die Attribute A_1 und A_2 für einen (inneren) Equi-Join identifiziert, wobei diese Bedingung im STAR-Schema (**Abbildung 4.1**) durch die Fremdschlüsselbeziehung zwischen A_1 und A_2 gegeben ist, und weiterhin erfüllt Attribut A_3 die Attributwertkardinalitätsbedingung, dann kann ein Bitmap-Join-Index vorberechnet werden. Dabei wird ein Bitmap-Index auf der Relation R über das Attribut A_3 , welches von Relation S referenziert wird, angelegt, wobei die Attribute A_1 und A_2 dann als Vergleichs- oder Bedingungswerte für die Join-Vorbereitung genutzt werden. In SQL kann diese Index-Erstellung wie folgt aussehen.

```
CREATE BITMAP INDEX RS
ON Relation R (Relation S.Attribut A3)
FROM Relation R, Relation S
WHERE Relation R.Attribut A1=Relation S.Attribut A2;
```

Dadurch wird deutlich, daß der (innere) Equi-Join mit Hilfe des Indexes und dem Attribut A_3 umgesetzt wird, und so die geringe Kardinalität des Wertebereiches von A_3 ausgenutzt werden kann. Anschließend lassen sich die Relationen nun durch einen natürlichen Verbund (NATURAL JOIN) unter Nutzung des erstellten Bitmap-Indexes zusammenfügen. Die Umsetzung stellt sich dann wie folgt dar.

```
SELECT * FROM
Relation S NATURAL JOIN Relation R
WHERE Attribut A3='...';
```

Durch diese Vorberechnung der Verbundattribute kann der Aufwand für Anfragen diesen Typs durch die Reduzierung der Join-Kosten stark verringert werden, da ohne diesen Bitmap-Join-Index von Oracle folgendes SQL-Statement bei jeder Anfrage durchgeführt werden müßte.

```
SELECT * FROM
Relation R, Relation S
WHERE Relation R.Attribut A1=Relation S.Attribut A2
AND Attribut A3='...';
```

Es läßt sich daraus schließen, daß dieser Effekt bei mehr als zwei Verbundpartnern, was in einem STAR-Schema und Data-Warehouse-Anwendungen nicht unüblich ist, eine stärkere Bedeutung zugesprochen werden muß. Aber auch diese Herangehensweise hat ihre Nachteile, denn das Update-Problem für Bitmap-Indexe konnte auch durch diesen Ansatz nicht umgangen werden, da bei Änderungen von Attribut A_3 in Relation S auch der Bitmap-Join-Index reorganisiert werden muß (siehe **Abschnitt 2.2**). Eine weitere Problematik offenbart sich, wenn Anfragen gestellt werden, bei denen weitere Kriterien oder Bedingungen in der WHERE-Klausel eingefügt werden, denn dann kann Oracle den vorher erstellten Bitmap-Join-Index nicht verwenden. Ob es eine Möglichkeit gibt den Index für einen Teil der Anfrage zu nutzen, ist bis dato offen. Die von Oracle vorgestellten Bitmap-Join-Indexe unterscheiden sich, wie zuvor beschrieben, stark von der allgemeinen Definition der Join-Indexe, die ebenfalls in den vorherigen Abschnitten betrachtet wurden. Beide Arten von Bitmap-Join-Indexen sollen in die weiteren Ausführungen einbezogen werden.

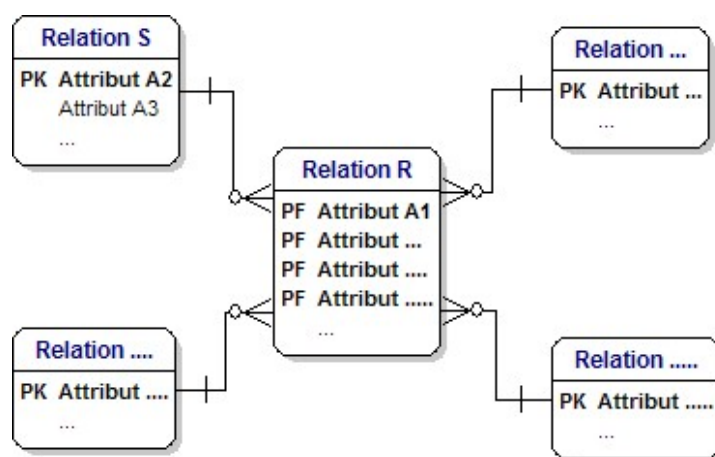


Abbildung 4.1: STAR-Schema für Oracle-Bitmap-Join-Index

4.4 Entscheidungsmodell für Bitmap-Index Selektion

Der folgende Abschnitt wird sich weiteren Neuerungen im speziellen Bezug auf Bitmap-Indexe widmen, und dabei Vorschläge erläutern, die die bisher genannten Probleme mit Bitmap-Index-Strukturen versuchen zu lösen.

Zuerst wird das in **Abschnitt 2.2** beschriebene Update-Problem diskutiert, und ein Vorschlag für die Reduzierung dieser Problematik erläutert. Wie bereits bekannt, sind die Reorganisationskosten für einen Bitmap-Index durch die notwendigen Matrixoperationen von quadratischem Aufwand ($O(n^2)$), wodurch die Durchführung der Reorganisation gerade bei größeren Datenmengen wie in OLAP-Systemen oder Data-Warehouses zu einer unpraktikablen Situation führt. Da die Reorganisation zwar mit Tricks wie der Hilfsspalte beim Löschen von Tupeln nicht bei jeder Änderung durchgeführt werden muß, aber diese dadurch nur hinausgezögert wird, ergibt sich daraus keine befriedigende Lösung.

Denn es kann keine Abschätzung darüber getroffen werden, ob diese Verzögerung ausreicht, um eine praktikable Lösung mit Hilfe von Bitmap-Indizes anbieten zu können, da trotz dieser Bemühungen ein zu häufig auftretendes Update-Verhalten nicht beachtet wird.

Daher soll mit Hilfe dieses Ansatzes das Update-Verhalten schon bei der Bestimmung von Index-Kandidaten für das Datenbankmanagementsysteme abgeschätzt werden. Dazu sollen die vorgestellten Kostenmodelle aus **Abschnitt 4.1** zur Hilfe herangezogen und erweitert werden. Wie bereits erwähnt, war der Grundgedanke den Profit eines empfohlenen Indexes durch die Differenz aus den Kosten einer Anfrage Q_k und den Kosten für die Ausführung dieser Anfrage unter Zuhilfenahme dieses bestimmten Indexes I_i zu bilden. Im Allgemeinen soll die Abschätzung des Update-Verhaltens durch die Nutzung von Dichtefunktionen von Wahrscheinlichkeiten gegeben sein, wobei diese dann durch den Datenbankadministrator oder das Datenbankmanagementsystem je nach Anwendungsfall spezifiziert werden kann. Durch diese Spezifizierung kann eine allgemein gültige Vorschrift für die Empfehlung von Bitmap-Indizes erreicht werden, da sich die Dichtefunktion einer Zufallsvariable X je nach Workload oder Verhalten des Datenbanksystems anpassen läßt. So ist es zum Beispiel möglich das Szenario abzubilden, bei dem die Workload-Analysen oder Heuristiken des Datenbankadministrators aussagen, daß die Wahrscheinlichkeit für eine Update-Operation auf einem Attribut steigt, je länger keine solche Operation durchgeführt wurde. Oder andersherum gesehen, wenn ein Attribut vor kurzer Zeit geändert wurde, dann ist die Wahrscheinlichkeit gering, daß dies zu einem nahen Zeitpunkt wieder geschehen wird. Dieses Anwendungsbeispiel kann mit Hilfe einer Exponentialverteilung dargestellt werden, die wie folgt definiert ist.

$$f_\lambda(x) = \lambda e^{-\lambda x} \text{ für } x \geq 0 \text{ und } f_\lambda(x) = 0 \text{ für } x < 0$$

Für die Berechnung des Profits eines Indexes I_i und die endgültige Abbildung des oben genannten Anwendungsfalles muß die Wahrscheinlichkeit für ein Update von der maximal erreichbaren Wahrscheinlichkeit, die im Allgemeinen *eins* beträgt, abgezogen werden. Die Begründung für diese Vorschrift liegt einerseits darin, daß, falls die Änderungswahrscheinlichkeit eines Attributes *null* beträgt, dementsprechend der Faktor für den Profit des Indexes I_i *eins* betragen sollte. Weiterhin kann so mit Hilfe der Exponentialverteilung erreicht werden, daß das oben beschriebene Update-Verhalten abgebildet wird, da die Dichtefunktion der Exponentialverteilung genau das Gegenteil zeichnet. Also ergibt sich in diesem speziellen Fall für die Parametrisierung α des Profits die folgende Vorschrift.

$$\alpha = 1 - f_\lambda(x)$$

Diese Betrachtungen werden verallgemeinert und auf weitere Wahrscheinlichkeitsfunktionen übertragen, da die Bildung des Parameters α von allgemeiner Gültigkeit ist. Daraus läßt sich schließen, daß die allgemeine Berechnungsformel wie folgt aussieht.

$$\alpha = 1 - \Psi$$

Dabei steht der Parameter Ψ allgemein für eine beliebige Dichtefunktion einer Wahrscheinlichkeit, und wird die Unabhängigkeit des Parameters α von einer bestimmten Dichtefunktion erhalten. Weiterhin ist es möglich jede Dichtefunktion durch ein Parameter zu spezifizieren, wie es durch den Parameter λ bei der Exponentialverteilung

möglich ist. Dadurch kann zum Beispiel unter Verwendung einer gewichteten Normalverteilung erreicht werden, einen engen Zeitrahmen abzubilden, in dem alle Änderungen am Datenbanksystem stattfinden. Dies kann der Fall sein, wenn jeweils nach Ablauf so genannter Epochen¹⁰ Daten in das System integriert werden, wodurch sich spezielle Strategien ableiten lassen. Eine dieser Strategien wäre es, keine Änderungen an den Indexen, die durch die Datenintegration notwendig wären, durchzuführen, und diese an Stelle dessen zu löschen und neu zu erstellen.

Dieser Ansatz soll nun in die zuvor dargelegten Betrachtungen zur Berechnung des Profits für einen Index und Index-Konfigurationen eingebettet werden. Am sinnvollsten scheint es zu sein, dies direkt für die grundlegenden Gedanken aus **Abschnitt 3.1** durchzuführen, da sich auf diese Art und Weise die Veränderungen durch alle darauf aufbauenden Betrachtungen übertragen. Demzufolge sollte der Parameter α in die erste Betrachtung des Profits eines Indexes I_j für eine Anfrage Q_k einbezogen werden. Damit ist die Gewichtung des Profits eines Indexes durch den Parameter α gegeben, und stellt sich wie folgt dar.

$$profit(Q_k, I_i) = \alpha * max \{0, cost(Q_k) - cost(Q_k, I_i)\} \text{ alternativ}$$

$$profit(Q_k, I_i) = (1 - \Psi) * max \{0, cost(Q_k) - cost(Q_k, I_i)\}$$

Mit Hilfe dieser Herangehensweise ist es nun möglich, daß Update-Verhalten von Indexen sowie Index-Konfigurationen abzuschätzen, da sich diese auf Basis der Profitabschätzungen der einzelnen Indexe zusammensetzen (siehe **Abschnitt 3.1**). Dabei ist zu erwähnen, daß nicht für jeden Index eine separate Abschätzung notwendig sein muß, da es ohne Weiteres möglich ist, diese für eine Menge von Indexen oder eine Index-Konfiguration zu approximieren. Dies kann für die Gruppierung von ähnlichem Verhalten von Indexen genutzt werden, wobei es zusätzlich eine Reduzierung der Komplexität und somit des Aufwandes mit sich bringt. Daraus läßt sich der Schluß ziehen, daß zugleich die Genauigkeit der Abschätzungen sinkt, aber diese gegebenenfalls nicht in einem solch starken Maße gefordert ist. So könnte auch eine durchschnittliche Abschätzung (Approximation) für die Gesamtmenge von Bitmap-Indexen erfolgsversprechend sein.

Der eben vorgestellte Ansatz war eine Einschränkung des Gewinns von Bitmap-Index-Strukturen, um so ihr schlechtes Update-Verhalten abbilden zu können. Der folgende Vorschlag wird sich dagegen auf eine der Stärken von Bitmap-Indexen beziehen, um diesen bisher unbeachteten Vorteil für genauere Abschätzungen verwenden zu können. Diese Betrachtungen werden sich auf die logische Verknüpfbarkeit von Bitmap-Indexen beziehen, und versuchen verschiedene Blickwinkel aufzuzeigen. Die Ausnutzung der logischen Verknüpfbarkeit setzt voraus, daß die erstellten Indexe atomar sind, das heißt sich auf ein Attribut beziehen. Der Grund dafür läßt sich dadurch erklären, daß zusammengesetzte Indexe oder Mehrkomponenten-Bitmap-Indexe diese Eigenschaft verlieren oder diese stark eingeschränkt wird. Die Ursache dafür liegt darin, daß bei zusammengesetzten Bitmap-Indexen und Varianten von ihnen der mögliche Suchraum für Verknüpfungen reduziert wird. Denn die Teile der entstandenen Indexe können nicht mehr, wie in **Abschnitt 4.2** gezeigt, für andere Verknüpfungen genutzt werden, da es sich die Präfix-Eigenschaften nicht nutzen lassen. Daher muß an dieser Stelle die Entscheidung getroffen werden, welche der Eigenschaften für den Anwendungsfall mehr Erfolg versprechen. Demzufolge wird die Verknüpfbarkeit entweder eingeschränkt, oder diese

¹⁰Einteilung in Zeitabschnitte (Dauer)

soll gerade ausgenutzt werden.

Wie im vorherigen Ansatz soll die mögliche Ausnutzung eines Indexes für Teilanfragen einer, ihm nicht zugeordneten, Anfrage durch eine Wahrscheinlichkeit dargestellt werden, das heißt der Index wurde nicht für oder durch diese Anfrage erzeugt. Auf diese Art und Weise ist es möglich die Wiederverwendung des Indexes (Verknüpfbarkeit) durch Statistiken des Datenbankmanagementsystems oder Heuristiken eines Datenbankadministrators abzuschätzen. Dazu sei wie zuvor eine beliebige Dichtefunktion Θ gegeben, die die Wahrscheinlichkeit für die gegebene Situation ausdrücken kann. An dieser Stelle wird es sich unter den gegebenen Voraussetzungen weniger um Verteilungen wie die Exponentialverteilung handeln, da diese ungeeignet erscheinen, um durchschnittliche Erwartungen abzuschätzen. Näher liegen für diesen Sachverhalt Verteilungen wie die Student t- oder Normalverteilung, die wiederum durch Parametrisierung exakter definiert werden können. Gerade die Student t-Verteilung scheint dafür geeignet zu sein, da sich diese für Stichprobenanalysen als besonders geeignet erwiesen hat, und ist wie folgt definiert.

$$f_n(x) = \frac{\Gamma(\frac{n+1}{2})}{\sqrt{n\pi}\Gamma\frac{n}{2}} \left(1 + \frac{x^2}{n}\right)^{-\frac{n+1}{2}} \text{ für } -\infty < x < +\infty$$

$$\text{mit } \Gamma(x) = \int_0^{+\infty} t^{x-1} e^{-t} dt$$

Dabei ist zu beachten, daß die zu erwartenden Wahrscheinlichkeiten relativ klein sein werden, denn es ist nachzuvollziehen, daß die Wiederverwendung von Indexen nicht mit einer ähnlichen hohen Häufigkeit auftritt, wie der Index an sich verwendet wird. Wenn eine solche Abschätzung mit Hilfe einer Wahrscheinlichkeit wie der Student t-Verteilung durchgeführt wurde, dann kann diese als zusätzlicher Gewinn in den Profit eingerechnet werden. Es sei dazu gegeben, daß ein Faktor ω genau dann *eins* ist, wenn die Wahrscheinlichkeit der Wiederverwendung *null* beträgt. Daraus kann geschlußfolgert werden, daß die Wiederverwendungswahrscheinlichkeit den Wert *eins* erhöhen muß, um als Gewinn betrachtet werden zu können. Dies führt für die Berechnung des Parameters ω zu folgender Vorschrift.

$$\omega = 1 + f_n(x)$$

Im Allgemeinen wurde eine Menge von Dichtefunktionen als Θ bestimmt, wodurch die Betrachtungen für die Student t-Verteilung ohne Beschränkungen der Allgemeinheit angewendet werden können, und zu folgender Formel führen.

$$\omega = 1 + \Theta$$

Dieser Sachverhalt kann nun in die Ansätze zur Berechnung des Profits eines Indexes mit einbezogen werden, und auf diese Art und Weise wird die Abschätzung des tatsächlichen Gewinns gegebenenfalls exakter als zuvor bestimmt. Dabei fließt der Faktor ω in die, wie im vorherigen Ansatz verwendete, Gleichung mit ein, und ergibt dann wie folgt.

$$profit(Q_k, I_i) = \omega * \max \{0, cost(Q_k) - cost(Q_k, I_i)\} \text{ alternativ}$$

$$profit(Q_k, I_i) = (1 + \Theta) * \max \{0, cost(Q_k) - cost(Q_k, I_i)\}$$

Auch für diesen Lösungsansatz gilt die allgemeine Verwendbarkeit für alle darauf aufbauenden Betrachtungen, wie es für den Lösungsvorschlag zuvor festgestellt werden konnte.

Eine Reduzierung der Komplexität ist auch an dieser Stelle durch eine Verallgemeinerung der Dichtefunktion Θ für eine Gruppe von Indexen oder eine Approximation für eine Index-Konfiguration möglich, welche durch eine Abwägung der Genauigkeitsansprüche festgelegt werden kann.

Die abschließende Aufgabe dieses Abschnittes ist es, zu klären wie beide Lösungsvorschläge für die zukünftigen Index-Empfehlungen vereint werden können. Dabei muß berücksichtigt werden, daß der abgeschätzte Gewinn für eine Wiederverwendung nur einen Teil der Profitabschätzungen eines Indexes ausmacht, wobei sich die Parametrisierung aufgrund des bekannten Update-Problems auf den Gesamtprofit an sich auswirkt. Daraus kann gefolgert werden, daß die Abschätzung für das Update-Verhalten die des zusätzlichen Profits einschließen wird, um so eine genauere Gesamtbetrachtung der Bitmap-Indexe und deren Eigenschaften zu erhalten. Weiterhin wurden die Vorschriften unabhängig von Komplexitätsreduktionen oder Genauigkeitskriterien definiert, so daß sie im allgemeinsten Sinne als Basis für spätere Spezialisierungen im Sinne eines Anwendungsfalles genutzt werden können. Zusammenfassend ergeben sich daraus also folgende Gleichungen.

$$profit(Q_k, I_i) = \alpha(\omega * max\{0, cost(Q_k) - cost(Q_k, I_i)\}) \text{ alternativ}$$

$$profit(Q_k, I_i) = (1 - \Psi)((1 + \Theta) * max\{0, cost(Q_k) - cost(Q_k, I_i)\})$$

Beide Gleichungen zeigen wie zuvor beschrieben die Einbindung der Parameter α und ω . Dabei ist zu erkennen, daß der Parameter ω direkt auf die maximale Differenz der Kostenbetrachtung aus $cost(Q_k)$ und $cost(Q_k, I_i)$ auswirkt, und diese in Abhängigkeit von der Dichtefunktion Θ mindestens mit einem Wert von *eins* gewichtet wird. Durch das Auftreten von Wiederverwendungen des Indexes I_i wird der Nutzen dieses Indexes um den Faktor ω erhöht. Für den Parameter α dagegen wirkt sich die Gewichtung für die Berechnung des Profits $profit(Q_k, I_i)$ global aus, das heißt es sind alle vorherigen Teilschritte oder Ergebnisse von der Gewichtung des Parameters α betroffen. Die zugehörige Dichtefunktion Ψ bildet dabei das Update-Verhalten des Indexes I_i ab, worin sich der globale Einfluß begründet, da sich die Reorganisationskosten des Indexes I_i auf die gesamte Kostenbetrachtung auswirken. Denn es ist nachzuvollziehen, daß die Kosten einer Reorganisation sich stets negativ auswirken, und der Parameter α in Abhängigkeit von Ψ maximal einen Wert von *eins* erreichen kann.

4.5 Funktionsweise und Einbettung

Dieser Abschnitt stellt den Bezug zwischen den bisherigen Betrachtungen und Vorschlägen zu deren Nutzung und Einbettung in aktuelle Datenbankmanagementsysteme her. Dazu sollen verschiedene Blickwinkel betrachtet sowie analysiert werden.

Die durchgeführten Analysen beruhen wie zuvor beschrieben auf früheren Ansätzen, und werden diese erweitern. Dabei wurde darauf geachtet, eine von der Implementierung oder Einbettung in ein Datenbanksystem unabhängige Analyse sowie ein unabhängiges Entscheidungsmodell zu entwerfen. Daraus ergibt sich, daß ein Entwickler oder eine Firma die Entscheidung treffen kann, ob die Optimierungen des Datenbanksystems (in diesem Fall Index-Self-Tuning) in einem separaten Tool durchgeführt werden sollen, oder aber in das bisherige Konstrukt eines Optimierers des Datenbankmanagementsystems eingebunden wird. Im Folgenden werden beide Varianten der Umsetzung diskutiert.

Als erste Variante wird die vollständige Einbettung der neuen Optimierungsvorschläge in ein bestehendes Datenbankmanagementsystem diskutiert, das heißt sie sind Bestandteil dieser Software. Dies hat zur Folge, daß die einzelnen Analyse- und Optimierungsschritte in die jeweiligen Komponenten des Datenbankmanagementsystems eingebunden werden müssen. Dazu gehört, das Monitoring nutzen zu können, und dieses gegebenenfalls zu modifizieren, sowie die Möglichkeit anzubieten Statistiken zu erstellen und auszuwerten. Weiterhin ist es notwendig die Empfehlungen durch die Erweiterung eines so genannten Optimierers für das Datenbankmanagementsystem oder -administrator nutzbar zu machen, denn ohne eine integrierte Empfehlung der Neuerungen würden diese für die Anfrageanalyse ignoriert werden. Dafür ist es notwendig die neuen Kostenmodelle und Index-Strukturen in den Index-Wizard oder Optimierer einzupflegen, und diese gegenüber bisherigen Strukturen zu bewerten. Weiterhin muß die Möglichkeit gegeben sein, daß die bisherigen Schritte (wie in **Abschnitt 4.1** beschrieben) wiederholt werden können. Für die Erstellung von Indexten können wie bei den vorherigen Schritten die Grundkonzepte eines Datenbankmanagementsystems genutzt und falls nötig angepaßt werden. Dies kann zur Folge haben, daß Index-Strukturen nicht nur erweitert werden, sondern ebenso Neue in das Datenbankmanagementsystem eingeführt werden müssen. Die Umsetzung dieser Variante bedarf daher einer genaueren Aufwandsanalyse, die dem Anwendungsfall entsprechend bewertet werden kann. Der Vorteil dieser Methode liegt darin, daß sehr große Teile der bestehenden Techniken eines Datenbankmanagementsystems nutzbar sind, und die Erweiterungen speziell auf dieses Datenbankmanagementsystem (Alteration) zu geschnitten werden können. Dies ist im Sinne des Software-Engineering sinnvoll, da an dieser Stelle sowohl Techniken des Re-Engineering [SPL03] als auch des Requirement-Engineering [HK83] genutzt werden können. Allerdings ergründet sich aus dem Vorteil der optimalen und performanten Anpassung an ein Datenbankmanagementsystem ebenso der Nachteil, daß die Anpassungen für jedes oder gegebenenfalls auch für unterschiedliche Generationen eines Datenbankmanagementsystems durchgeführt werden müssen.

Bei der zweiten Umsetzungsvariante dagegen werden die Erweiterungen oder Neuentwicklungen als eigenständiges Tool zur Unterstützung des Datenbankmanagementsystems umgesetzt. Dazu werden alle Neuerungen innerhalb des zu entwickelnden Tools implementiert, wobei dies, die vom Datenbankmanagementsystem zur Verfügung gestellten Funktionen nutzt. Dabei werden die bereitgestellten Funktionen gegebenenfalls erweitert oder neu implementiert, falls diese für die gewünschte Funktionalität nicht geeignet sind. Daraus resultiert, daß das Datenbankmanagementsystem an sich unberührt bleibt, und dadurch keine Probleme mit Lizenzen oder Zugriff auf geschützte Software entstehen können. Weiterhin kann auf diese Art und Weise eine Unabhängigkeit vom jeweiligen Datenbankmanagementsystem erreicht werden, denn das Tool bietet alle nötigen Funktionen für die Neuerungen. Diese können mit Hilfe einer definierten Schnittstelle zwischen dem Tool und dem Datenbanksystem (sowie dessen Funktionalität) zugreifen, wodurch es nicht weiter nötig ist die Erweiterungen für jedes Datenbankmanagementsystem anzupassen. Die Erweiterungen können dadurch allgemein gültig definiert werden, und alle Anpassungen an ein bestimmtes Datenbanksystem finden innerhalb der definierten Schnittstelle statt, was dazu führt, daß die Wiederverwendung der Funktionalität des Tools und damit Implementierungen (Code) stark erhöht werden können. Dadurch kann der Aufwand für die Einbettung der Vorschläge in kommerzielle Datenbankmanagementsysteme stark reduziert werden, denn die Implementierung neuer Funktionalität findet

einmalig für das Tool statt. Die Definition der entsprechenden Schnittstellen für das Tool kann mit Hilfe der Anbieter von Datenbankmanagementsystemen durchgeführt werden, denn nur an dieser Stelle müssen systemspezifische Komponenten gehandhabt werden. Allerdings hat diese Vereinfachung bei der Entwicklung und Anpassung an ein Datenbanksystem die Folge, daß das Tool und dessen Funktionalität nur eingeschränkt für ein bestimmtes Datenbankmanagementsystem optimiert werden können.

Beide Varianten weisen ihre Vor- und Nachteile auf, wobei zu klären ist, ob der nötige Aufwand für die explizite Anpassung an ein Datenbankmanagementsystem gerechtfertigt ist. Andererseits ist es zu prüfen, ob ein Verzicht auf Optimalität der Erweiterungen akzeptiert werden kann, sowie die Möglichkeit der Verwaltung verschiedenster Schnittstellen für die Datenbankmanagementsysteme. In beiden Fällen ist die Zustimmung und Unterstützung der jeweiligen Datenbankmanagementsystemhersteller zu überprüfen, wodurch die Entscheidung für eine der Variante mit hoher Wahrscheinlichkeit beeinflußt wird.

Kapitel 5

Evaluierung

Das folgende Kapitel ist der Evaluierung der vorherigen Analysen und Vorschläge gewidmet, um so die Richtigkeit der zuvor getroffenen Schlußfolgerungen zu beweisen. Dazu wird zwischen den einzelnen Aspekten unterschieden, die in dieser Arbeit analysiert wurden. Zunächst sollen Aussagen zur Verwendung von Bitmap-Indexen als Such-Index getroffen werden, woraufhin ein Vergleich der Effektivität von verschiedenen Join-Indexen durchgeführt wird. Zuletzt werden die Ergebnisse zusammenfassend in Bezug zu den möglichen Anwendungsgebieten gesetzt.

Dabei werden alle Analysen mit den in **Abschnitt 2.6** beschriebenen Techniken durchgeführt. Die dafür genutzte Plattform war ein Sun Blade 1500 (Silver) mit einem 1,5 Gigahertz (**GHz**) UltraSPARC-IIIi Prozessor und 2048 Megabyte (**MB**) Random Access Memory (**MB**). Weitere Informationen können bei der Firma Sun nachgelesen werden [SM06].

5.1 Zugriffs- und Such-Bitmap-Index gegenüber B/B⁺-Baum

Für die folgenden Betrachtungen wird ein einfaches Schema (siehe **Abbildung 5.1**) genutzt, bei dem mit Hilfe von Index-Strukturen die Anfragen beschleunigt werden sollen. Das Schema soll eine repräsentative Berechnung in einem Data-Warehouse abbilden, in dem der Einfachheit halber ein trivialer Geschäftsprozeß dargestellt sein soll. Dazu werden die Ein- und Ausgaben sowohl in Bezug zur Zeit als auch zu einer Einheit gesetzt. Daraus lassen sich wirtschaftliche Rückschlüsse wie Kosten oder Gewinn ziehen, wie sie typisch für Analyse in einem Data-Warehouse sind. Im Speziellen wurde an dieser Stelle eine Anfrageform gewählt, bei der die Daten zu einem so genannten Cube [Leh03, 105,288] aggregiert werden. Zur Gegenüberstellung der verschiedenen Index-Strukturen wurden jeweils die gleichen Rahmenbedingungen für die Berechnung der Aggregation gewählt, wobei einzig und allein die genutzte Index-Strukturen gewechselt wurde. Die gewählte Anfrage für das repräsentative Beispiel lautet dann wie folgt.

```
SELECT Gruppenname, Tag, SUM(Umsatz) AS Einnahmen
FROM Einnahmen NATURAL JOIN Zeit NATURAL JOIN Einheit
GROUP BY CUBE (Gruppenname, Tag);
```

Mit dieser Anfrage werden die Einnahmen einer Gruppe, die jeweils einem Tag zugeordnet sind, mit Hilfe der Aggregation des Umsatzes $SUM(Umsatz)$ berechnet, sowie nach Gruppenname und Tag gruppiert. Der inhaltliche Aspekt des Beispiels wird nicht weiter von Bedeutung sein, da dieser nur zur Verdeutlichung des realen Anwendungsfalles erläutert wurde. Für den Vergleich der Index-Strukturen werden im Folgenden die

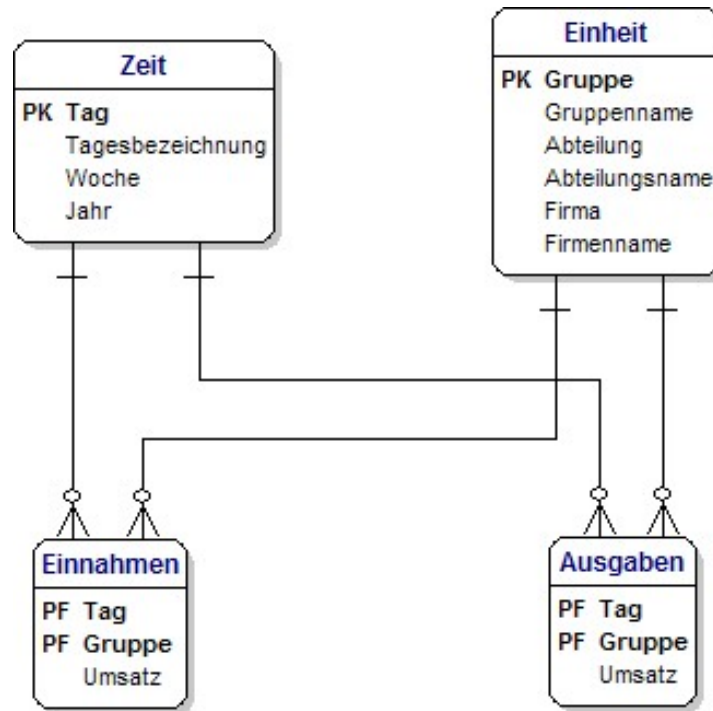


Abbildung 5.1: Verwendetes Schema für die Cube-Berechnung

Werte für die *CPU Cost* herangezogen, dabei handelt es sich um einen normierten Wert aus den Statistiken des Datenbankmanagementsystems. Dieser wird durch den Operator *EXPLAIN PLAN* während der Laufzeit der entsprechenden Anfrage mit Hilfe der Statistiken des Datenbankmanagementsystems von Oracle gewonnen, und kann somit ohne Einschränkung der Allgemeinheit für die Vergleiche genutzt werden. Weiterhin soll der, in den vorherigen Kapiteln erwähnte, Speicherplatzbedarf der Index-Strukturen verglichen werden, dieser wird mit Hilfe der Statistikfunktionen von Oracle ermittelt.

Zunächst wurde die oben vorgestellte Anfragen ohne Hilfe von speziellen Indexen für das Fallbeispiel durchgeführt, dabei konnte der Optimierer des Datenbankmanagementsystems nur die von ihm selbst automatisch erzeugten Indexe für die Primärschlüssel (siehe **Abschnitt 2.1**) der einzelnen Tabellen genutzt werden. Diese Durchführung der zuvor beschriebenen Anfrage wird im Folgenden als die Variante *ohne Index* angesehen. Weiterhin werden für die folgenden Betrachtungen der verschiedenen Index-Strukturen jeweils drei Indexe angelegt, wobei es sich um Indexe auf den Tabellen *Zeit*, *Einnahmen* und *Einheit* des Beispiels aus der **Abbildung 5.1** handelt. Diese Modifikation wird im späteren Verlauf als Variante *B-Baum* bezeichnet werden. Wie bei anderen Datenbankmanagementsystemen auch nutzt Oracle B-Bäume als Standard-Indexstruktur, wodurch keine explizite Angabe der Indexstruktur bei der Indexerzeugung notwendig ist. Diese werden mit Hilfe der Structured Query Language (**SQL**) unter Verwendung einer B-Baum-Indexstruktur erzeugt, und stellen sich wie folgt dar.

```

CREATE INDEX Zeit_Index
  ON Zeit
    (Tag,Woche,Jahr);
CREATE INDEX Fakten_Index
  ON Einnahmen
    (Umsatz,Gruppe,Tag);
CREATE INDEX Einheit_Index
  ON Einheit
    (Firmenname,Firma,Abteilungsname,Abteilung,Gruppenname,Gruppe);

```

Als letzte Abwandlung für das Fallbeispiel (**Abbildung 5.1**) werden die Bitmap-Index-Strukturen zum Einsatz kommen. Dabei ist allerdings zu beachten, daß die im Datenbankmanagementsystem von Oracle eine andere Version von Bitmap-Indexten verwendet wird als die in **Abschnitt 2.2** beschriebene Variante. In der Version von Oracle sind nur die äußeren Blattknoten als Bitvektoren organisiert, und in den Hierarchieebenen darüber handelt es sich um Baumstrukturen wie dem B-Baum (siehe Abschnitt 2.3). Für die folgenden Betrachtungen soll dieser Fakt keinen weiteren Einfluß haben, da es sich um eine Abschwächung der Bitmap-Eigenschaften handelt. Dabei muß beachtet werden, daß das genutzte Datenbankmanagementsystem von Oracle das einzige kommerzielle System ist, daß Bitmap-Index-Strukturen unterstützt. Somit kann das verwendete System trotz der Variation als Referenz für die weiteren Betrachtungen angesehen werden. Unter den gleichen Bedingungen, wie sie für Variante *B-Baum* beschrieben wurden, können die Indexte für die Variante *Bitmap-Index* wie folgt erzeugt werden.

```

CREATE BITMAP INDEX Zeit_Index
  ON Zeit
    (Tag,Woche,Jahr);
CREATE BITMAP INDEX Fakten_Index
  ON Einnahmen
    (Umsatz,Gruppe,Tag);
CREATE BITMAP INDEX Einheit_Index
  ON Einheit
    (Firmenname,Firma,Abteilungsname,Abteilung,Gruppenname,Gruppe);

```

Für die folgende Analyse wurden zwei Operationen ausgewählt, bei denen der Einfluß der Index-Strukturen nachzuweisen war. Dabei handelt es sich um den Zugriff auf die Tabelle *Einheit* und dem *Merge-Join* zwischen den Tabellen *Einheit* und *Zeit*.

Wie in **Tabelle 5.1** zu erkennen ist, konnten die Kosten für die Selektion auf der Tabelle *Einheit* durch den *Index-Scan* mit beiden Index-Strukturen deutlich gesenkt werden. Denn ohne die Verwendung von Index-Strukturen betragen die CPU-Kosten (*CPU Cost*) 57687, wogegen diese mit Hilfe eines B-Baum-Indexes auf 26321 gesenkt werden konnten. Die Erstellung des B-Baum-Indexes führte somit bereits zu einer Halbierung der CPU-Kosten (*CPU Cost*) bei der Durchführung der repräsentativen Anfrage. Allerdings konnte für das in **Abbildung 5.1** Szenario die Leistung durch die Verwendung eines Bitmap-Indexes weiter gesteigert werden, denn die CPU-Kosten (*CPU Cost*) reduzierten sich von 26321 mit einem B-Baum weiter auf 17929 mit Hilfe eines Bitmap-Indexes. Daraus ist zu erkennen, daß CPU-Kosten (*CPU Cost*) weiter um circa 1/3 gegenüber der *B-Baum*-Variante gesunken sind, und die Kosten (*CPU Cost*) eines Bitmap-Index

Indexstruktur	Index-Scan	Merge-Join
ohne Index	57687	37745244
B-Baum	26321	37713879
Bitmap-Index	17929	37705486

Tabelle 5.1: CPU Costs der einzelnen Indexvarianten

nur noch 1/3 der ursprünglichen Kosten von 57687 betragen. Zusätzlich können der **Tabelle 5.1** die Kosten für den *Merge-Join* zwischen den Tabellen *Einheit* und *Zeit* entnommen werden.

Die Unterschiede zwischen den einzelnen Varianten sind an dieser Stelle nicht so drastisch, wie sie es für die Selektion auf der Tabelle *Einheit* waren, da die Index-Strukturen nur einen geringen Einfluß auf die Durchführung des *Merge-Join* haben. Der Grund dafür liegt darin, daß die Daten in der Tabelle *Zeit* so organisiert sind, daß der Index des Primärschlüssels für eine schnelle Sortierung in alle drei Varianten verwendet wird. Aber trotz dieses Fakts ist die Verwendung von Index-Strukturen zu erkennen, denn die Differenz der Kosten (*CPU Cost*) zwischen der Verwendung von B-Baum-Indizes (37713879) und keinen Indizes (37745244) beträgt trotzdem 31365. Wie zuvor bei der Selektion von Daten läßt sich durch einen Bitmap-Index (37705486) diese Differenz auf 39758 vergrößern, was sich die Einsparung der Kosten bei der Selektion auf der Tabelle *Einheit* verdeutlichen läßt. Dieser Vorteil wirkt sich wie trivial nachzuvollziehen ist ebenfalls für den *Merge-Join* aus, da dieser die Daten für die Verbindung der Daten ebenfalls selektieren muß. Die Unterschiede lassen sich durch die Berechnung der Differenzen nachvollziehen. Die Differenzen für die Selektion zwischen den Index-Strukturen und der Variante *ohne Index* betragen 31366 für die *B-Baum* Variante und 39758 für die des *Bitmap-Index*. Diese Werte entsprechen sehr genau den Gewinnen der Index-Strukturen bei der Durchführung des *Merge-Join*, denn diese betragen 31365 für den *B-Baum* beziehungsweise 39758 für den *Bitmap-Index*. Dadurch wird gezeigt wie wichtig für die Betrachtungen einer Index-Empfehlung und deren Nutzen die Einbeziehung der Wiederverwendung von Indizes ist, denn an dieser Stelle handelt es sich nur um ein triviales Beispiel für die Erklärungen und ein Beweis der Aussagekraft, der zuvor getroffen Aussagen.

Weiterhin ist während der Analyse der verschiedenen Anfragevarianten aufgefallen, daß bei der Nutzung von Indizes jeweils nur Ein-/Ausgabe-Kosten (*I/O Costs*) von *eins* entstanden, wogegen für die Variante *ohne Index* bei jeder Durchführung *I/O Cost* von *drei* entstanden. Dieser Fakt kann für weitere Betrachtungen von höherer Bedeutung sein, als es an dieser Stelle für den Nachweis der Rentabilität von Bitmap-Indizes der Fall war. Zusätzlich bleibt zu erwähnen, daß für die Variante *B-Baum* bei der ersten Durchführung ein weiterer Index durch den Optimierer von Oracle angelegt wurde. Da die Erstellungskosten nur einmalig anfielen, wurden sie der Verständlichkeit halber aus den Betrachtungen genauso herausgelassen, wie den Nutzen dieses Indexes aus den Betrachtungen für die *B-Baum*-Variante herauszurechnen, da diese das Ergebnis nur verbessern könnte. Des Weiteren wird auf diese Art und Weise das Argument geschwächt, daß das gewählte Szenario ausschließlich für Bitmap-Index-Strukturen gewählt wurde, und diese somit bevorteilen würde.

Ein weiterer Aspekt ist der Speicherplatzbedarf der Index-Strukturen, allerdings muß die Variante *ohne Index* an dieser Stelle außen vor gelassen werden, da es weder Nutzen noch

Speicherplatzbedarf für Indexe gibt. Auf den ersten Blick erscheinen alle Indexe, egal welcher Indexstruktur sie angehören, gleich groß, da für alle angelegten Indexe aufgrund der geringen Datenmengen und die, vom Datenbankmanagementsystem als Standardwert gesetzte, Ausgangsgröße von 64 Kilobyte (**KB**) nicht überschritten wird. Um den tatsächlichen Speicherplatzbedarf der jeweiligen Indexstruktur zu ermitteln, muß dieser mit Hilfe der Anzahl der Datensätze (Tupel) berechnet werden. Für einen B-Baum wird dazu der Speicherplatz für den Tupelidentifikator jedes Tupels bestimmt, und man erhält durch die Anzahl der Tupel den mindestens benötigten Speicherplatz. Allerdings handelt es sich dabei nur um eine Abschätzung. Denn durch die Zuordnung der Datensätze auf Seiten (**siehe Abschnitt 2.2**) kann der benötigte und der tatsächliche Speicherplatzbedarf sich unterscheiden, aber auf diese Art und Weise kann die minimale Zahl benötigter Seiten errechnet werden. Die Anzahl hängt dabei von der Seitengröße (Vielfaches der Blockgröße des Datenträgers) des Index-Pools ab, die je nach Anwendungsszenario gewählt werden kann. Dagegen wird für einen Bitmap-Index die Anzahl der Werte eines Attributs in Bits berechnet, und man erhält den nötigen Speicherplatz. Die auf diese Art

Indexstruktur	Einheit-Index	Einnahmen-Index	Zeit-Index
Bitmap-Index in KB	0,012	38,496	0,401
B-Baum in KB	0,375	1231,875	12,832

Tabelle 5.2: Speicherplatzbedarf von Index-Strukturen

und Weise errechneten Werte für das repräsentative Beispiel (**Abbildung 5.1**) sind in **Tabelle 5.2** dargestellt. Wie deutlich zu erkennen ist, bestätigen diese Werte die Behauptungen des **Abschnittes 2.2**, denn der Speicherplatzbedarf der Bitmap-Indexe ist deutlich geringer als der Speicherbedarf der B-Bäume. Weiterhin ist zu erkennen, daß das Verhältnis zwischen dem Speicherbedarf des Bitmap-Indexes und B-Baumes nahezu konstant ist. Werden die einzelnen Verhältnisse gebildet, so erhält man aus 0,375 und 0,012 den Wert 31,25 oder aus 1231,875 und 38,496 den Wert 32, und ebenso ergibt 12,832 durch 0,401 den Wert 32. Daraus kann für das gewählte Beispiel geschlußfolgert werden, daß die Bitmap-Indexe für dieses Szenario um mehr als das 30-fache kleiner sind als die B-Bäume mit denselben Daten. Aufgrund dieser Eigenschaft der Bitmap-Indexe ist es möglich, daß bei gleicher Index-Pool-Größe wesentlich mehr Bitmap-Indexe als B-Bäume verwaltet und genutzt werden können.

Die letzten Betrachtungen werden kurz die Gesamtbearbeitungszeiten darlegen, die für das genutzte Beispiel entstanden sind. Dazu wurden zunächst die Verarbeitungszeiten des in **Abschnitt A.1** eingefügten Skriptes, sowie die Anfragezeiten der verwendeten Testanfrage verglichen. Die Testanfrage dazu lautet wie folgt.

```
SELECT * FROM Einheit WHERE Gruppe = 1;
```

Die **Tabelle 5.3** zeigt die gemessenen Werte und verdeutlicht, daß die Erstellung beziehungsweise Nutzung von Indexen nur einen sehr geringen Anteil an den Kosten oder dem Nutzen für die Skriptausführung haben. Dies läßt sich an den sehr ähnlichen Verarbeitungszeiten für alle drei Ausführungsvarianten erkennen. Der Unterschied zu der Ausführung *ohne Index* beträgt gerade einmal 0,3 Prozent für die *B-Bäume* und 2,04 Prozent für die *Bitmap-Indexe*. Die Erklärung dafür liegt darin, daß die Indexe auf die Erstellung von Tabellen oder anderen Datenbankobjekten keinen Einfluß nehmen. Dies

kann erst der Fall sein, wenn die Indexe und Tabellen bereits existieren, und diese für Aggregationen von Objekten genutzt werden können. Deutlicher wird ein Unterschied

Indexstruktur	Skriptausführung	Testanfrage
Bitmap-Index in Sek.	284,020	0,0820
B-Baum in Sek.	289,032	0,2376
ohne Index in Sek.	289,95	0,2026

Tabelle 5.3: Verarbeitungszeiten des verwendeten Skriptes und einer Testanfrage

in der Anfragezeit der verwendeten Testanfrage sichtbar. Für die Varianten *ohne Index* und *B-Baum* ist der Unterschied schon 14,5 Prozent, wobei an dieser Stelle die *B-Baum*-Variante schlechter abschneidet. Der Grund dafür kann allerdings in der geringen Datenmenge liegen, da unter diesen Umständen ein *FULL TABLE SCAN* schneller sein kann. Trotzdem ist der Unterschied zu der Ausführung mit *Bitmap-Indexten* sehr deutlich, denn für diese Ausführung beträgt die benötigte Zeit nur 40,5 Prozent von der Ausführungszeit der Variante *ohne Index*. Dies kann nicht durch die geringe Datenmenge begründet werden, da ansonsten die *Bitmap-Indexte* durch den Index-Zugriff schlechter abschneiden müßten. Also beweisen die Messungen auch an dieser Stelle, daß die Verwendung von Bitmap-Indexten erfolgversprechend ist.

Der Schluß aus den Beobachtungen und getroffenen Feststellungen ist es, daß es ein praktikabler Ansatz ist, Bitmap-Index-Strukturen unter den in den vorherigen Kapiteln besprochenen Rahmenbedingungen einzusetzen. Denn sie können auf diese Art und Weise die Performance für das Suchen von Daten und das Zugreifen diese bisheriger Index-Strukturen verbessern, und somit zu einer zufriedenstellenderen Lösung in verschiedenen Anwendungsfällen beitragen. Durch die repräsentative Analyse für Such- und Zugriffsindexe auf Bitmap-Basis kann an dieser Stelle gesagt werden, daß die Betrachtungen der Kosten für die Ausführung von Anfragen positiv für die Verwendung von Bitmap-Indexten sprechen.

5.2 Analyse von Bitmap-Join-Indexten und STAR-Joins

Im Rahmen dieses Abschnittes sollen die erlangten Erkenntnis in Bezug zu (Bitmap-) Join-Indexten erörtert und nachgewiesen werden.

Für die Evaluierung wurde verschiedenste Szenarien entwickelt, die aber verworfen wurden, da es für diese keine Unterstützung für die Verwendung von Bitmap-Join-Indexten gab. Alle Schemata hatten die Gemeinsamkeit, daß diese den grundlegenden Aufbau eines STAR-Schemas nutzen. Weiterhin wurden verschiedene Datenmengen, Attributwertkardinalitäten sowie Schlüsselbeziehungen verwendet, aber für alle Varianten blieb das Ergebnis gleich. Dies lautete, daß der Optimierer des Datenbankmanagementsystems die erstellten Bitmap-Join-Indexte ignorierte. Nach einiger Nachbearbeitung wurden die so genannte Optimizer-Hints [Mei05] von Oracle verwendet, die den Optimierer derart beeinflussen sollten, daß dieser die angegebenen Zugriffspfade oder Indexte nutzt. Allerdings wurden auch diese Optimizer-Hints vom Datenbankmanagementsystem ignoriert, denn sie konnten an der Art der Ausführung oder Nutzung von zum Beispiel vorgegebe-

nen Indexen nichts ändern.

Daraufhin wurde das von Oracle verwendete Beispiel [Bur02] analysiert und nachempfunden, das für die Vorstellung des Konzeptes der Oracle-Bitmap-Join-Indexe genutzt wurde. Die **Abbildung 5.2** zeigt das übernommene Beispiel, und im **Anhang A.2** befindet sich das entsprechende Skript. Der nötige Bitmap-Join-Index kann demzufolge mit Hilfe des nachfolgenden Syntax erstellt werden.

```
CREATE BITMAP INDEX Bauteil_Lieferant_Land
ON Lager (B.Bauteil_Typ, LI.Land)
FROM Lager LA, Bauteil B, Lieferant LI
WHERE LA.Bauteil_ID = B.Bauteil_ID
AND LA.Lieferanten_ID = LI.Lieferanten_ID;
```

Entsprechend dem vorgestellten Konzept der Bitmap-Join-Indexe von Oracle (siehe **Abschnitt 4.3**) wurde die verwendete Anfrage für dieses Beispiel wie folgt entworfen.

```
SELECT /*+ INDEX(LAGER) [BAUTEIL_LIEFERANT_LAND] */ Name_Lieferant
FROM Bauteil NATURAL JOIN Lager NATURAL JOIN Lieferant
WHERE Bauteil_Typ = 'achtundachzig' AND Land = 'Deutschland';
```

Der mit `/*` und `*/` umschlossene Syntax realisiert die Verwendung eines Optimizer-Hints, und soll die Nutzung des Indexes *BAUTEIL-LIEFERANT-LAND* auf der Tabelle *Lager* für diese Anfrage erzwingen. Aber trotz zahlreicher Versuche und Konsultationen in einschlägigen Foren und Dokumentationen erbrachte auch dieses Beispiel mit der Nutzung der Optimizer-Hints nicht den gewünschten Erfolg. Es gibt keine klare Begründung für

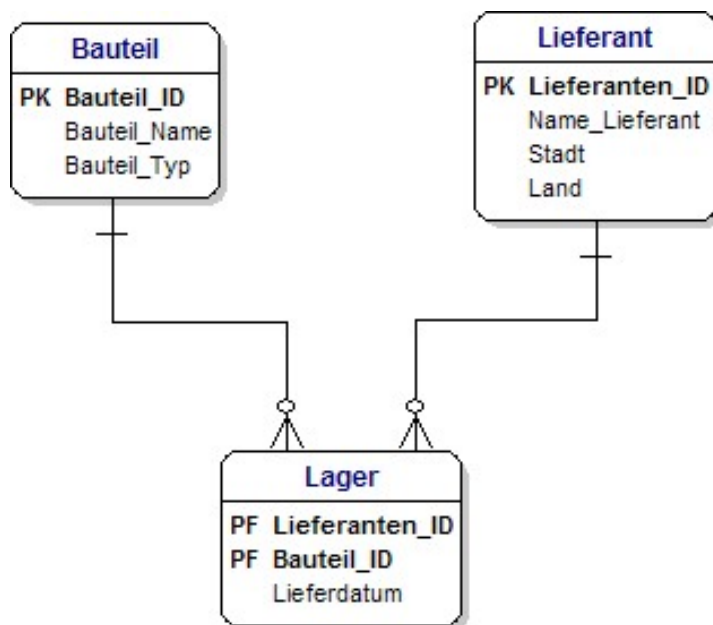


Abbildung 5.2: Nach empfundenes Beispiel für Bitmap-Join-Indexe

diesen Zustand, denn es wurden alle Parameter eingehalten, und anhand der Dokumentation von Oracle umgesetzt. Auch weitere Analysen wie zum Beispiel die Nutzung von Statistikfunktionen des Datenbankmanagementsystems führte nicht zum Ziel. Aufgrund

dieser Tatsachen war es nicht möglich Vergleichswerte für die Analyse eines Fallbeispiels zu gewinnen.

Zusätzlich muß an dieser Stelle erwähnt werden, daß ohne eine Verwendung von Indexten für die Variante *ohne Indexte* ebenfalls keine Vergleichswerte für die Evaluierung gesammelt werden konnte. Die B-Bäume dagegen mußten entfallen, da es keinerlei Unterstützung seitens des Datenbankmanagementsystems für Join-Indexte auf B-Baumbasis gibt. Bei einem Versuch diese Art von Index zu erstellen, wurde dies mit der Fehlermeldung „*Es handelt sich um keinen Bitmap-Index, verwenden sie das Keyword BITMAP*“ quittiert, wodurch für die Variante *B-Baum* ebenfalls keine Werte gemessen werden konnten. Daraus kann der Schluß gezogen werden, daß nur Bitmap-Join-Indexte als Join-Indexte unterstützt werden, aber die Verwendung dieses Konzeptes bisher nur unzureichend unterstützt wird.

5.3 Anwendung und Ergebnis

Am Ende dieses Kapitels werden die Vorschläge und Analysen zusammenfassend dargestellt, um so die möglichen Anwendungsgebiete aufzuzeigen. Dabei sollen die Veränderungen zur bisherigen Situation beschrieben werden, und somit die Erweiterungen und ihr Nutzen abgewägt werden können.

Die Verwendung von Bitmap-Index-Strukturen ist in heutigen kommerziellen Datenbankmanagementsystemen wenig oder gar nicht unterstützt. Der Grund dafür liegt in der aufwendigen Beurteilung der Rentabilität dieser Indexstruktur, welche durch die in dieser Arbeit gewonnenen Erkenntnisse erleichtert werden sollen. Denn bisher lag die Entscheidung überwiegend bei den Datenbankadministratoren, was die Analyse und Entscheidungsphase bisher sehr personalaufwendig und kostenintensiv machte. Erschwerend dazu kamen die in den vorherigen Kapitel beschriebenen Eigenschaften der Bitmap-Indexte, die eine allgemeingültige Nutzung dieser Struktur nicht zulassen. Aber aufgrund der weiteren Eigenschaften und deren Auswirkungen, die zuvor ausführlich analysiert und diskutiert wurden, konnte nachgewiesen werden, daß die Bitmap-Indexte in Gebieten des Data-Warehousings und andere Bereichen, die ähnliche Rahmenbedingungen besitzen, sehr effizient arbeiten.

Motiviert durch diese Rahmenbedingungen, war es das Ziel die Bitmap-Indexte praktikabel in einer Vielzahl von Szenarien einsetzen zu können. Dazu wurde bewährte Vorschläge wie von Oracle genutzt, aber diese bezogen sich allein auf die Entwurfsphase oder manuell angestoßene Analyse so genannter Index-Wizards. Allerdings konnten auf diese Art und Weise Erfahrungen genutzt werden, um diese aufwendige Lösung zu verbessern, und mit deren Hilfe eine Unterstützung für Bitmap-Indexte im Bereich des Self-Tunings eines Datenbanksystems zu schaffen. Dazu wurde in **Abschnitt 3.1** beschrieben, wie der Nutzen eines Indexes im Allgemeinen bestimmt werden kann. Daraufhin wurden in **Kapitel 4** Kostenmodelle für die Berechnung der Kosten-/Nutzen-Analyse im Sinne des Self-Tunings vorgestellt. Weiterhin wurden verschiedene Teilaspekte in diese Betrachtungen mit einbezogen, die einen Einfluß auf den tatsächlich Gewinn eines Indexes haben. Am schwerwiegendsten war dabei das zuvor beschriebene Update-Problem abschätzen zu können, da dieses das härteste Kriterium gegen die Nutzung von Bitmap-Indexten ist. Denn die Handhabung von Updates ist bei dieser Art von Index-Strukturen derart aufwendig, daß auch der größte Nutzen eines Indexes durch häufiges Update-Verhalten

zu Nichte gemacht wird.

Daher war die Nutzung von Bitmap-Index bisher ausschließlich auf Anwendungen beschränkt, bei denen das Update-Verhalten vernachlässigt werden konnte. Diese ist für einige Data-Warehouse-Lösungen praktikabel, aber nicht in allen Szenarien anwendbar. Denn bei dieser Art von Data-Warehouse-Anwendungen gibt es Zeiten, in denen die Datenbestände aktualisiert werden, aber die Indexe nicht aktualisiert sondern neuerstellt werden. Durch die Entwicklung neuer Kostenmodelle und die Einbettung der Update-Problematik in die Nutzenanalyse kann nun eine Abschätzung für das Update-Verhalten getroffen werden. Dadurch wurde die Möglichkeit geschaffen, die Bitmap-Indexe nicht nur im Bereich des Data-Warehousing ohne Updates, sondern auch in neuen allgemeineren Bereichen nutzen zu können. Somit sollen die Vorteile der Bitmap-Indexe auch in bisher für diese Art der Index-Strukturen unbeachtete Bereiche angewandt werden.

Die auf diese Art und Weise gewonnenen Möglichkeiten für die Nutzung von Bitmap-Indexen erweitern den Suchraum für die Optimierung um neue Gesichtspunkte, wodurch für die Bereiche des Index- und Self-Tunings neue Zusammenhänge entstehen. Diese sind durch bisherigen Techniken eines Design Advisors oder Self-Tuning-Tools nutzbar, dadurch können verbesserte Lösungen für bisherige Problemstellungen gefunden werden, aufgrund der Erweiterung der Möglichkeiten durch neue Eigenschaften. Das neue Potential wurde in den vorherigen Abschnitten analysiert, und zeigt die Relevanz der vorgeschlagenen Erweiterungen. Allerdings ist zu beachten, daß auch weiterhin die Bitmap-Indexe andere Index-Strukturen nicht ersetzen können werden. Denn durch die Entwicklung der Kostenmodelle und den dazugehörigen Betrachtungen ist klar, daß der Einsatz und die Abschätzung für Bitmap-Indexe verbessert werden konnte. Trotz alledem sind sie auch künftig nur performant einsetzbar, wenn es sich um Szenarien handelt, in denen wenige Änderungen der Daten stattfinden. Allerdings kann durch die vorher getroffenen Erkenntnisse das bisherige Spektrum derart erweitert werden, daß Bitmap-Indexe wesentlich allgemeiner im Sinne des Anwendungsszenarios nutzbar sind.

Kapitel 6

Zusammenfassung

Das abschließende Kapitel dieser Arbeit wird die bisher gewonnenen Erkenntnisse zusammenfassen und noch einmal kurz darstellen. Daraufhin folgt ein Ausblick, der die weiterführenden Entwicklungen, Analysen und gegebenenfalls anwendungsspezifische Studien vorstellen soll.

Motiviert wurde diese Arbeit durch die aktuelle Situation im Bereich der Optimierung von Datenbanksystemen, und diese Möglichkeiten zu erweitern. Dabei wurde speziell Bezug auf das Index-Tuning und dessen stetige und autonome Erweiterung dem Self-Tuning genommen. Aufgrund der ständig wachsenden Anforderung an aktuelle Datenbanksysteme wurde das Ziel verfolgt, bisher nicht betrachtete Aspekte des Index-Tunings zu analysieren und anwendbar zu machen. Für die Erweiterung der bisherigen Ansätze wurden Bitmap-Indexe gewählt, daß diese bisher nicht in aktuelle Forschungen im Bereich des Self-Tunings einbezogen wurden. Der Grund dafür liegt im schlechten Update-Verhalten dieser Index-Strukturen, denn ein Update macht eine Reorganisation mit quadratischem Aufwand notwendig. Dies ist bedingt durch einen schwer einzuschätzenden Gewinn dieser Indexe, wodurch diese Art der Index-Strukturen nur in wenigen Szenarien (meist mit speziellen Updatephasen) eingesetzt wurden.

Für weitere Analysen war es nötig eine grundlegende Kosten-/Nutzen-Funktion aufzustellen, um so zunächst eine Abschätzung für einen Index (-kandidat) durchführen zu können. Die nötige Herangehensweise und das Ergebnis wurden in **Abschnitt 3.1** erläutert. Diese einfache Berechnung des Profit eines Indexes zu einer Anfrage ($profit(Q_k, I_i)$) wurde unter den gegebenen Rahmenbedingungen wie Speicherplatzbedarf $size(I_i)$ und Einbeziehung der Verwaltungskosten $mcost(I_i)$ auf eine Menge von Indexen (Index-Konfiguration C) erweitert. Auf diese Art und Weise konnte der Gesamtgewinn einer Index-Konfiguration C für eine Menge von Anfragen Q_m bestimmt und maximiert werden. Aufbauend auf dieses grundlegendem Verständnis für eine Kosten-/Nutzen-Analyse (Gewinn) eines Indexes wurden verschiedene bisher bekannte Ansätze im Bereich des Self-Tunings vorgestellt und angewendet. Dabei wurde für die folgenden Betrachtungen dieser Arbeit der vorgestellte Regelkreis des Index-Self-Tunings (**Abbildung 3.1**) für exemplarische Erläuterungen genutzt, doch lassen sich die Betrachtungen und Schlußfolgerungen ohne Beschränkung der Allgemeinheit auf weitere Ansätze übertragen.

Anschließend widmete sich das vierte Kapitel der Entwicklung eines Konzeptes für das Self-Tuning von Bitmap-Indexen. Zuerst beschrieb der **Abschnitt 4.1** die grundlegenden Probleme wie die Komplexität des Index Selection Problems. Dabei wurde auf ver-

schiedene kritische Gesichtspunkte eingegangen, die eine dynamische Lösung des Index Selection Problems ermöglichen oder dessen Komplexität reduzieren. Im Detail wurden die einzelnen Schritte (Observation, Prediction und Reaction) des Regelkreises für das Index-Self-Tuning betrachtet. Die Analyse der einzelnen Schritte führte zu einer objektiven Diskussion verschiedener Kriterien, wodurch diese für jeden Anwendungsfall erneut bewertet werden müssen. Denn es handelt sich dabei um die Abschätzungen inwieweit die Optimalität als totale Grenze gesehen werden muß, oder ob diese durch Einschränkungen der Genauigkeit in ihrer Komplexität begrenzt werden kann oder darf. Im weiteren Verlauf konnten Index-Kandidaten und die gegebenen Rahmenbedingungen im Zusammenhang bewertet, wodurch eine detaillierte Analyse möglich wurde. Dabei wurde auf verschiedene Probleme des autonomen (dynamischen) Index-Self-Tuning eingegangen, die zu verschiedenen Bewertungskriterien des Profits von Index-Empfehlungen führten. Die unterschiedlichen Aspekte der Beurteilung des Gewinns von Indexen wurden dabei formalisiert, und dienen als Grundlage der weiteren Betrachtungen.

Darauf aufbauend wurden Methoden und Kriterien für die Anfrageanalyse (**Abschnitt 4.3**) vorgestellt. Diese sind notwendig, um Index-Empfehlungen und deren Profitabschätzungen erstellen zu können. Dazu wurde die Attributwertkardinalitätsbedingung eingeführt, die ein allererstes Kriterium für die Wahl von Bitmap-Indexen darstellt. Denn wird diese Bedingung nicht erfüllt, besagen zahlreiche Heuristiken und Fallstudien, daß die Verwendung von Bitmap-Indexen wenig profitversprechend sein wird. Mit Hilfe dieses Kriteriums und den zuvor aufgestellten Kosten-/Nutzen-Analysen konnten bereits Empfehlungen für Zugriffs- und Such-Bitmap-Indexe ausgesprochen werden, wobei die Erfüllung der Attributwertkardinalitätsbedingung zur Profitbestimmung dieser Indexe führte. Um die Vollständigkeit der Analyse von Bitmap-Indexen zu gewährleisten, mußten auch die Bitmap-Join-Indexe in die Betrachtungen der Anfrageanalyse aufgenommen werden. Diese wurde in vier unterschiedlichen Fällen betrachtet, die alle die Gemeinsamkeit hatten, daß die Attributwertkardinalitätsbedingung ein Bestandteil der Voraussetzungen war. Der Unterschied zwischen den einzelnen Fällen liegt dabei in der Erfüllung oder Nichterfüllung der Attributwertkardinalitätsbedingung der einzelnen Join-Partner. Die vier Fälle überdecken dabei den gesamten möglichen Suchraum für diese Art von Indexen und deren Kombinationen, und die Lösung dieser ergab in der Gesamtheit eine Regelmenge für die Abschätzung einer Bitmap-Join-Index-Empfehlung. Die Regelmenge kann dabei ohne weitere Einschränkungen auf eine beliebige Zahl von Verbundpartnern erweitert werden, dazu muß die Regelmenge in Abhängigkeit von der Anzahl der Partner unter Verwendung der aufgestellten Regeln erweitert werden. Ausgehend von diesen Betrachtungen ist es möglich eine unter verschiedenen Kriterien bewertete Anfrageanalyse für Bitmap-Index-Strukturen durchzuführen.

Im **Abschnitt 4.4** wurde auf die erschwerenden Eigenschaften der Bitmap-Indexe eingegangen. Dabei war es das Ziel Lösungen für das bekannte Update-Problem und die Wiederverwendbarkeit von Indexen zu finden. Die Idee dabei war es die Wiederverwendbarkeit eines Indexes durch die Analyse des Workloads abzuschätzen, indem mit Hilfe von Beobachtungen des vergangenen Verhaltens des Datenbanksystems eine Wahrscheinlichkeitsfunktion bestimmt wird. Diese Aufgabe kann wie bisher auch ein Datenbankadministrator übernehmen, wodurch die Möglichkeit gegeben ist die gewonnenen Konzepte in verschiedenen Entwicklungsstufen zu einem autonomen (dynamischen) Index-Tuning zu überführen. Diese Herangehensweise für die Beurteilung eines zukünftigen Verhaltens des Datenbanksystems konnte weitgehend für die Abschätzung des Update-Verhaltens

übernommen werden. Dazu war es nötig die Wahrscheinlichkeitsfunktionen an diese Gegebenheiten anzupassen, und dabei wurden erneut die Beobachtungen des Workloads genutzt, um somit Vorhersagen für die zukünftige Entwicklung des Workloads treffen zu können.

Zum Abschluß des vierten Kapitels wurden die Zusammenhänge für eine Einbettung der neuen Konzepte in ein kommerzielles Datenbankmanagementsystem und deren Anwendung in der Praxis diskutiert. Dabei wurden zwei gegensätzliche Varianten besprochen. Zunächst wurde die Möglichkeit erörtert, die Erweiterung dieser Arbeit für das Index-Self-Tuning in ein bestehendes Datenbankmanagementsystem und dessen Optimierungsmechanismen zu integrieren. Im Gegensatz dazu konnte mit der zweiten Variante die Verwendung eines eigenständigen Tools angestrebt werden, wobei auch an dieser Stelle, soweit wie möglich, gegebene Mechanismen des Datenbankmanagementsystems genutzt werden sollten. Die Betrachtungen führten zu dem Schluß, daß die Entscheidung für eine der Möglichkeiten sehr stark vom jeweiligen Anwendungsfall und der Unterstützung der Hersteller von Datenbankmanagementsystemen abhängt.

Als Abschluß der wissenschaftlichen Analysen dieser Arbeit wurde im fünften Kapitel versucht die erlangten Erkenntnisse und Konzepte zu evaluieren. Dazu wurden zunächst mit Hilfe des in **Abschnitt 5.1** vorgestellten Beispiels die Zugriffs- und Such-Bitmap-Indexe untersucht. Dabei konnte festgestellt werden, daß sich Bitmap-Indexe sehr gut für Anwendungsfälle mit geringem Update-Verhalten eignen, solange die in den vorherigen Kapiteln beschriebenen Bedingungen eingehalten werden.

Weiterhin wurde die Nutzbarkeit von Bitmap-Join-Indexen in **Abschnitt 5.2** untersucht. Dabei traten schwerwiegende Probleme auf, die trotz zahlreicher Bemühungen nicht gelöst werden konnten. Dazu gehörte auch, daß B-Bäume für diese Betrachtungen seitens des Datenbankmanagementsystems außen vor bleiben mußten. Weiterhin konnten nur für die Variante *ohne Indexe* Messungen durchgeführt werden, aber diese haben keine Aussagekraft, wenn von keiner der beiden Index-Strukturen Vergleichswerte gewonnen werden konnten. Die fehlenden Meßwerte für die Bitmap-Join-Indexe resultieren aus der Problematik, daß diese Indexe in einer Vielzahl von Szenarien und deren Varianten nicht angewendet werden konnten. Auch die von Oracle angebotenen Funktionen für das Eingreifen in die Anfrageverarbeitung brachten nicht die Lösung dieses Problems. Als Schluß dieser fehlgeschlagenen Beweisführung kann nur darauf hingewiesen werden, daß Bitmap-Indexe auch weiterhin nur wenig oder gar nicht unterstützt werden.

Der **Abschnitt 5.3** beinhaltet die Analyse der bisherigen Beobachtungen und deren Bezug zu möglichen Anwendungsgebieten. Dabei wurden die Zusammenhänge zwischen den neuen Konzepten und deren Möglichkeiten in Bezug auf denkbare Erweiterungen aktueller Datenbankmanagementsysteme hergestellt. Des Weiteren konnten zukünftige Möglichkeiten unter Zuhilfenahme der neuen Konzepte und Erweiterungen für das Index-Tuning beziehungsweise Index-Self-Tuning dargestellt werden.

Im Verlauf dieser Arbeit wurden neue Herangehensweisen und Konzepte für das Index-Self-Tuning von Bitmap-Index-Strukturen vorgestellt. Unter Zuhilfenahme von allgemeingültigen Analysen der Bitmap-Index-Strukturen konnten mögliche Anwendungsgebiete herausgearbeitet werden. Dabei wurden die Vorteile durch die grundlegenden Eigenschaften dieser Index-Strukturen mit Hilfe von verschiedenen Gesichtspunkten unter Beweis gestellt. Die detaillierte Analyse und Bewertung der neu entworfenen Kostenmodelle und erweiterte Anfrageanalyse konnte, aufbauend auf bisherigen Aspekten des Index-Self-Tunings, mit stichhaltigen Argumenten durchgeführt werden. Allerdings ist

es notwendig die neuen Vorschläge zu implementieren und in aktuelle Datenbankmanagementsysteme (oder in Form eines Tools) einzubetten, um auf diese Art und Weise die Praktikabilität der neuen Ansätze unter Beweis zu stellen. Daraufhin sind weitere Analyse oder Fallstudien nötig, um die getroffenen Schlußfolgerungen und insbesondere die Verwendung der Kostenmodelle und Anfrageanalyse in ihrer Beweiskraft zu verbessern. Nach Abschluß der weitreichenden Studien in verschiedenen Anwendungsfällen bleibt zu prüfen, wie die auch weiterhin bestehenden Probleme der Bitmap-Indexe verringert werden können, um so eine weitere Verbreitung dieser effektiven Index-Struktur für zahlreiche Anwendungsbereiche zu fördern. Denn der Grund für die geringe Nutzung von Bitmap-Indexten liegt in sehr großem Maße in der Problematik des Update-Verhaltens dieser Indexe.

Anhang A

Anhang

An dieser Stelle wurden die verwendeten Skripte eingebunden, die für die verwendeten Beispiele genutzt wurden. Dadurch soll die Nachweisbarkeit gewährleistet werden. Zusätzlich können diese als Grundlage für weitere Arbeiten dienen.

A.1 Beispiel für Zugriffs- und Such-Bitmap-Indexe

Im Folgenden werden die Data Description Language (**DDL**) und Data Manipulation Language (**DML**) des Beispiels aus **Abbildung 5.1** dargestellt.

```
SET AUTOTRACE OFF
drop table plan_table;
drop dimension zeit_dim;
drop dimension einheits_dim;
drop table einnahmen;
drop table ausgaben;
drop table zeit;
drop table einheit;

create table zeit (
jahr number,
woche number,
tagesbezeichnung varchar(12),
tag number constraint pk_zeit primary key
);

begin
  for i in reverse 1 .. 365 loop
    insert into zeit values (1993, trunc(i/7),
      to_char(sysdate, 'HH24:MI:SS'), i);
    insert into zeit values (1994, trunc(i/7),
      to_char(sysdate, 'HH24:MI:SS'), i+365);
    insert into zeit values (1995, trunc(i/7),
      to_char(sysdate, 'HH24:MI:SS'), i+730);
```

```

insert into zeit values (1996, trunc(i/7),
  to_char(sysdate, 'HH24:MI:SS'), i+1095);
insert into zeit values (1997, trunc(i/7),
  to_char(sysdate, 'HH24:MI:SS'), i+1460);
insert into zeit values (1998, trunc(i/7),
  to_char(sysdate, 'HH24:MI:SS'), i+1825);
insert into zeit values (1999, trunc(i/7),
  to_char(sysdate, 'HH24:MI:SS'), i+2190);
insert into zeit values (2000, trunc(i/7),
  to_char(sysdate, 'HH24:MI:SS'), i+2555);
insert into zeit values (2001, trunc(i/7),
  to_char(sysdate, 'HH24:MI:SS'), i+2920);
end loop;
end;
/

create table einheit (
firmenname varchar(20),
firma number,
abteilungsname varchar(20),
abteilung number,
gruppenname varchar(20),
gruppe number constraint pk_einheit primary key
);

insert into einheit values ('MAPA', 1, 'Entwicklung', 1, 'Installation', 1);
insert into einheit values ('MAPA', 1, 'Entwicklung', 1, 'Business', 2);
insert into einheit values ('MAPA', 1, 'Vertrieb', 2, 'Logistik', 3);
insert into einheit values ('OSAP', 2, 'Entwicklung', 3, 'Installation', 4);
insert into einheit values ('OSAP', 2, 'Entwicklung', 3, 'Business', 5);
insert into einheit values ('OSAP', 2, 'Vertrieb', 4, 'Logistik', 6);
insert into einheit values ('OLAP', 2, 'Entwicklung', 3, 'Installation', 7);
insert into einheit values ('OLAP', 2, 'Entwicklung', 3, 'Business', 8);
insert into einheit values ('OLAP', 2, 'Vertrieb', 4, 'Logistik', 9);
insert into einheit values ('OLTP', 2, 'Entwicklung', 3, 'Installation', 10);
insert into einheit values ('OLTP', 2, 'Entwicklung', 3, 'Business', 11);
insert into einheit values ('OLTP', 2, 'Vertrieb', 4, 'Logistik', 12);
insert into einheit values ('OAPA', 1, 'Entwicklung', 1, 'Installation', 13);
insert into einheit values ('OAPA', 1, 'Entwicklung', 1, 'Business', 14);
insert into einheit values ('OAPA', 1, 'Vertrieb', 2, 'Logistik', 15);
insert into einheit values ('MSAP', 2, 'Entwicklung', 3, 'Installation', 16);
insert into einheit values ('MSAP', 2, 'Entwicklung', 3, 'Business', 17);
insert into einheit values ('MSAP', 2, 'Vertrieb', 4, 'Logistik', 18);
insert into einheit values ('MLAP', 2, 'Entwicklung', 3, 'Installation', 19);
insert into einheit values ('MLAP', 2, 'Entwicklung', 3, 'Business', 20);
insert into einheit values ('MLAP', 2, 'Vertrieb', 4, 'Logistik', 21);
insert into einheit values ('MLTP', 2, 'Entwicklung', 3, 'Installation', 22);

```

```
insert into einheit values ('MLTP', 2, 'Entwicklung', 3, 'Business', 23);
insert into einheit values ('MLTP', 2, 'Vertrieb', 4, 'Logistik', 24);
insert into einheit values ('MAPA', 1, 'Entwicklung', 1, 'Installation', 25);
insert into einheit values ('MAPA', 1, 'Entwicklung', 1, 'Business', 26);
insert into einheit values ('MAPA', 1, 'Vertrieb', 2, 'Logistik', 27);
insert into einheit values ('OSAP', 2, 'Entwicklung', 3, 'Installation', 28);
insert into einheit values ('OSAP', 2, 'Entwicklung', 3, 'Business', 29);
insert into einheit values ('OSAP', 2, 'Vertrieb', 4, 'Logistik', 30);
insert into einheit values ('OLAP', 2, 'Entwicklung', 3, 'Installation', 31);
insert into einheit values ('OLAP', 2, 'Entwicklung', 3, 'Business', 32);
insert into einheit values ('OLAP', 2, 'Vertrieb', 4, 'Logistik', 33);
insert into einheit values ('OLTP', 2, 'Entwicklung', 3, 'Installation', 34);
insert into einheit values ('OLTP', 2, 'Entwicklung', 3, 'Business', 35);
insert into einheit values ('OLTP', 2, 'Vertrieb', 4, 'Logistik', 36);
insert into einheit values ('OAPA', 1, 'Entwicklung', 1, 'Installation', 37);
insert into einheit values ('OAPA', 1, 'Entwicklung', 1, 'Business', 38);
insert into einheit values ('OAPA', 1, 'Vertrieb', 2, 'Logistik', 39);
insert into einheit values ('MSAP', 2, 'Entwicklung', 3, 'Installation', 40);
insert into einheit values ('MSAP', 2, 'Entwicklung', 3, 'Business', 41);
insert into einheit values ('MSAP', 2, 'Vertrieb', 4, 'Logistik', 42);
insert into einheit values ('MLAP', 2, 'Entwicklung', 3, 'Installation', 43);
insert into einheit values ('MLAP', 2, 'Entwicklung', 3, 'Business', 44);
insert into einheit values ('MLAP', 2, 'Vertrieb', 4, 'Logistik', 45);
insert into einheit values ('MLTP', 2, 'Entwicklung', 3, 'Installation', 46);
insert into einheit values ('MLTP', 2, 'Entwicklung', 3, 'Business', 47);
insert into einheit values ('MLTP', 2, 'Vertrieb', 4, 'Logistik', 48);
insert into einheit values ('MAPA', 1, 'Entwicklung', 1, 'Installation', 49);
insert into einheit values ('MAPA', 1, 'Entwicklung', 1, 'Business', 50);
insert into einheit values ('MAPA', 1, 'Vertrieb', 2, 'Logistik', 51);
insert into einheit values ('OSAP', 2, 'Entwicklung', 3, 'Installation', 52);
insert into einheit values ('OSAP', 2, 'Entwicklung', 3, 'Business', 53);
insert into einheit values ('OSAP', 2, 'Vertrieb', 4, 'Logistik', 54);
insert into einheit values ('OLAP', 2, 'Entwicklung', 3, 'Installation', 55);
insert into einheit values ('OLAP', 2, 'Entwicklung', 3, 'Business', 56);
insert into einheit values ('OLAP', 2, 'Vertrieb', 4, 'Logistik', 57);
insert into einheit values ('OLTP', 2, 'Entwicklung', 3, 'Installation', 58);
insert into einheit values ('OLTP', 2, 'Entwicklung', 3, 'Business', 59);
insert into einheit values ('OLTP', 2, 'Vertrieb', 4, 'Logistik', 60);
insert into einheit values ('OAPA', 1, 'Entwicklung', 1, 'Installation', 61);
insert into einheit values ('OAPA', 1, 'Entwicklung', 1, 'Business', 62);
insert into einheit values ('OAPA', 1, 'Vertrieb', 2, 'Logistik', 63);
insert into einheit values ('MSAP', 2, 'Entwicklung', 3, 'Installation', 64);
insert into einheit values ('MSAP', 2, 'Entwicklung', 3, 'Business', 65);
insert into einheit values ('MSAP', 2, 'Vertrieb', 4, 'Logistik', 66);
insert into einheit values ('MLAP', 2, 'Entwicklung', 3, 'Installation', 67);
insert into einheit values ('MLAP', 2, 'Entwicklung', 3, 'Business', 68);
insert into einheit values ('MLAP', 2, 'Vertrieb', 4, 'Logistik', 69);
```

```
insert into einheit values ('MLTP', 2, 'Entwicklung', 3, 'Installation', 70);
insert into einheit values ('MLTP', 2, 'Entwicklung', 3, 'Business', 71);
insert into einheit values ('MLTP', 2, 'Vertrieb', 4, 'Logistik', 72);
insert into einheit values ('MAPA', 1, 'Entwicklung', 1, 'Installation', 73);
insert into einheit values ('MAPA', 1, 'Entwicklung', 1, 'Business', 74);
insert into einheit values ('MAPA', 1, 'Vertrieb', 2, 'Logistik', 75);
insert into einheit values ('OSAP', 2, 'Entwicklung', 3, 'Installation', 76);
insert into einheit values ('OSAP', 2, 'Entwicklung', 3, 'Business', 77);
insert into einheit values ('OSAP', 2, 'Vertrieb', 4, 'Logistik', 78);
insert into einheit values ('OLAP', 2, 'Entwicklung', 3, 'Installation', 79);
insert into einheit values ('OLAP', 2, 'Entwicklung', 3, 'Business', 80);
insert into einheit values ('OLAP', 2, 'Vertrieb', 4, 'Logistik', 81);
insert into einheit values ('OLTP', 2, 'Entwicklung', 3, 'Installation', 82);
insert into einheit values ('OLTP', 2, 'Entwicklung', 3, 'Business', 83);
insert into einheit values ('OLTP', 2, 'Vertrieb', 4, 'Logistik', 84);
insert into einheit values ('OAPA', 1, 'Entwicklung', 1, 'Installation', 85);
insert into einheit values ('OAPA', 1, 'Entwicklung', 1, 'Business', 86);
insert into einheit values ('OAPA', 1, 'Vertrieb', 2, 'Logistik', 87);
insert into einheit values ('MSAP', 2, 'Entwicklung', 3, 'Installation', 88);
insert into einheit values ('MSAP', 2, 'Entwicklung', 3, 'Business', 89);
insert into einheit values ('MSAP', 2, 'Vertrieb', 4, 'Logistik', 90);
insert into einheit values ('MLAP', 2, 'Entwicklung', 3, 'Installation', 91);
insert into einheit values ('MLAP', 2, 'Entwicklung', 3, 'Business', 92);
insert into einheit values ('MLAP', 2, 'Vertrieb', 4, 'Logistik', 93);
insert into einheit values ('MLTP', 2, 'Entwicklung', 3, 'Installation', 94);
insert into einheit values ('MLTP', 2, 'Entwicklung', 3, 'Business', 95);
insert into einheit values ('MLTP', 2, 'Vertrieb', 4, 'Logistik', 96);

create table einnahmen (
umsatz number,
gruppe number,
tag number,
constraint fk_einnahmen_einheit foreign key (gruppe) references einheit(gruppe),
constraint fk_einnahmen_zeit foreign key (tag) references zeit(tag)
);

select * from einnahmen order by umsatz desc;

create table ausgaben (
umsatz number,
gruppe number,
tag number,
constraint fk_ausgaben_einheit foreign key (gruppe) references einheit(gruppe),
constraint fk_ausgaben_zeit foreign key (tag) references zeit(tag)
);
```

```
begin
  for j in reverse 1 .. 96 loop
    for i in reverse 1 .. 3285 loop
      insert into einnahmen values (((i-1)*6+j)*1000,j, i);
      insert into ausgaben values (((i-1)*6+j-1)*1000,j, i);
    end loop;
  end loop;
end;
/

create dimension zeit_dim
  level tag is zeit.tag
  level woche is zeit.woche
  level jahr is zeit.jahr
  hierarchy zeit_roll_up(
    tag child of
    woche child of
    jahr
  )
  attribute tag determines (tagesbezeichnung);

create dimension einheits_dim
  level gruppe is einheit.gruppe
  level abteilung is einheit.abteilung
  level firma is einheit.firma
  hierarchy einheit_roll_up(
    gruppe child of
    abteilung child of
    firma
  )
  attribute gruppe determines (gruppenname)
  attribute abteilung determines (abteilungsname)
  attribute firma determines (firmenname);

rem Pre
rem =====

rem Anführungsplaene ansehen
@utlxplan

rem Performance-Messung
SET TIMING ON;
select gruppenname,tag,sum(umsatz) as einnahmen
from einnahmen natural join zeit natural join einheit
group by cube(gruppenname, tag);
delete plan_table;
```

```
explain plan for select gruppenname,tag,sum(umsatz) as einnahmen
from einnahmen natural join zeit natural join einheit
group by cube(gruppenname, tag);
select
```

```
    substr (lpad(' ', level-1) || operation || '
(' || options || ')',1,30 ) "Operation",
    object_name "Object"
from plan_table
start with id = 0
connect by prior id=parent_id;
```

```
set pause on
set pause "das war eine normale anfrage ohne index"
select * from einheit where gruppe = 1;
set pause off
```

```
CREATE INDEX Zeit_INDEX
  ON zeit
    (tag,woche,jahr);
CREATE INDEX Fakten_INDEX
  ON einnahmen
    (umsatz,gruppe,tag);
CREATE INDEX Einheit_INDEX
  ON Einheit
    (firmenname,firma,abteilungsname,abteilung,gruppenname,gruppe);
```

```
select gruppenname,tag,sum(umsatz) as einnahmen
from einnahmen natural join zeit natural join einheit
group by cube(gruppenname, tag);
delete plan_table;
explain plan for select gruppenname,tag,sum(umsatz) as einnahmen
from einnahmen natural join zeit natural join einheit
group by cube(gruppenname, tag);
select
```

```
    substr (lpad(' ', level-1) || operation || '
(' || options || ')',1,30 ) "Operation",
    object_name "Object"
from plan_table
start with id = 0
connect by prior id=parent_id;
```

```
set pause on
set pause "das war eine anfrage mit b-baum-index"
select * from einheit where gruppe = 1;
set pause off
```

```
CREATE BITMAP INDEX Zeit_INDEX
  ON zeit
    (tag,woche,jahr);
CREATE BITMAP INDEX Fakten_INDEX
  ON einnahmen
    (umsatz,gruppe,tag);
CREATE BITMAP INDEX Einheit_INDEX
  ON Einheit
    (firmenname,firma,abteilungsname,abteilung,gruppenname,gruppe);

select gruppenname,tag,sum(umsatz) as einnahmen
from einnahmen natural join zeit natural join einheit
group by cube(gruppenname, tag);
delete plan_table;
explain plan for select gruppenname,tag,sum(umsatz) as einnahmen
from einnahmen natural join zeit natural join einheit
group by cube(gruppenname, tag);
select
  substr (lpad(' ', level-1) || operation || '
(' || options || ')',1,30 ) "Operation",
  object_name "Object"
from plan_table
start with id = 0
connect by prior id=parent_id;

set pause on
set pause "das war eine anfrage mit bitmap-index"
select * from einheit where gruppe = 1;
set pause off

SET TIMING OFF

drop index Zeit_INDEX;
drop index Fakten_INDEX;
drop index Einheit_INDEX;
drop table plan_table;
drop dimension zeit_dim;
drop dimension einheits_dim;
drop table einnahmen;
drop table ausgaben;
drop table zeit;
drop table einheit;
```

A.2 Beispiel für Bitmap-Join-Indexe und STAR-Joins

In diesem Abschnitt soll nur das zuletzt verwendete Beispiel, das dem aus der Oracle Dokumentation [Bur02] entspricht, für die Analysen dargestellt werden. Die grafische Darstellung dazu befindet sich im fünften Kapitel (**Abbildung 4.1**).

```
CREATE TABLE Bauteil (  
    Bauteil_ID VARCHAR2(15) NOT NULL,  
    Bauteil_Name VARCHAR2(40),  
    Bauteil_Typ VARCHAR2(40),  
    PRIMARY KEY (Bauteil_ID)  
);  
  
CREATE TABLE Lieferant (  
    Lieferanten_ID VARCHAR2(15) NOT NULL,  
    Name_Lieferant VARCHAR2(40),  
    Stadt VARCHAR2(40),  
    Land VARCHAR2(40),  
    PRIMARY KEY (Lieferanten_ID)  
);  
  
CREATE TABLE Lager (  
    Lieferanten_ID VARCHAR2(15) NOT NULL,  
    Bauteil_ID VARCHAR2(15) NOT NULL,  
    Lieferdatum VARCHAR(40),  
    PRIMARY KEY (Lieferanten_ID, Bauteil_ID)  
);  
  
ALTER TABLE Lager  
    ADD FOREIGN KEY (Bauteil_ID) REFERENCES Bauteil (Bauteil_ID);  
  
ALTER TABLE Lager  
    ADD FOREIGN KEY (Lieferanten_ID) REFERENCES Lieferant (Lieferanten_ID);  
  
BEGIN  
    FOR i IN 1 .. 100 LOOP  
        INSERT INTO Bauteil VALUES (trunc(i+89900),300,'eins');  
        INSERT INTO Bauteil VALUES (trunc(i+89600),299,'zwei');  
        INSERT INTO Bauteil VALUES (trunc(i+89300),298,'drei');  
        INSERT INTO Bauteil VALUES (trunc(i+89000),297,'vier');  
        INSERT INTO Bauteil VALUES (trunc(i+88700),296,'fünf');  
        INSERT INTO Bauteil VALUES (trunc(i+88400),295,'sechs');  
        INSERT INTO Bauteil VALUES (trunc(i+88100),294,'sieben');  
        INSERT INTO Bauteil VALUES (trunc(i+87800),293,'acht');  
        INSERT INTO Bauteil VALUES (trunc(i+87500),292,'neun');
```

```
INSERT INTO Bauteil VALUES (trunc(i+87200),291,'zehn');
INSERT INTO Bauteil VALUES (trunc(i+86900),290,'elf');
INSERT INTO Bauteil VALUES (trunc(i+86600),289,'zwölf');
INSERT INTO Bauteil VALUES (trunc(i+86300),288,'dreizehn');
INSERT INTO Bauteil VALUES (trunc(i+86000),287,'vierzehn');
INSERT INTO Bauteil VALUES (trunc(i+85700),286,'fünfzehn');
INSERT INTO Bauteil VALUES (trunc(i+85400),285,'sechzehn');
INSERT INTO Bauteil VALUES (trunc(i+85100),284,'siebzehn');
INSERT INTO Bauteil VALUES (trunc(i+84800),283,'achtzehn');
INSERT INTO Bauteil VALUES (trunc(i+84500),282,'neunzehn');
INSERT INTO Bauteil VALUES (trunc(i+84200),281,'zwanzig');
INSERT INTO Bauteil VALUES (trunc(i+83900),280,'einundzwanzig');
INSERT INTO Bauteil VALUES (trunc(i+83600),279,'zweiundzwanzig');
INSERT INTO Bauteil VALUES (trunc(i+83300),278,'dreiundzwanzig');
INSERT INTO Bauteil VALUES (trunc(i+83000),277,'vierundzwanzig');
INSERT INTO Bauteil VALUES (trunc(i+82700),276,'fünfundzwanzig');
INSERT INTO Bauteil VALUES (trunc(i+82400),275,'sechsundzwanzig');
INSERT INTO Bauteil VALUES (trunc(i+82100),274,'siebenundzwanzig');
INSERT INTO Bauteil VALUES (trunc(i+81800),273,'achtundzwanzig');
INSERT INTO Bauteil VALUES (trunc(i+81500),272,'neunundzwanzig');
INSERT INTO Bauteil VALUES (trunc(i+81200),271,'dreißig');
INSERT INTO Bauteil VALUES (trunc(i+80900),270,'einunddreißig');
INSERT INTO Bauteil VALUES (trunc(i+80600),269,'zweiunddreißig');
INSERT INTO Bauteil VALUES (trunc(i+80300),268,'dreiunddreißig');
INSERT INTO Bauteil VALUES (trunc(i+80000),267,'vierunddreißig');
INSERT INTO Bauteil VALUES (trunc(i+79700),266,'fünfunddreißig');
INSERT INTO Bauteil VALUES (trunc(i+79400),265,'sechsunddreißig');
INSERT INTO Bauteil VALUES (trunc(i+79100),264,'siebenunddreißig');
INSERT INTO Bauteil VALUES (trunc(i+78800),263,'achtunddreißig');
INSERT INTO Bauteil VALUES (trunc(i+78500),262,'neununddreißig');
INSERT INTO Bauteil VALUES (trunc(i+78200),261,'vierzig');
INSERT INTO Bauteil VALUES (trunc(i+77900),260,'einundvierzig');
INSERT INTO Bauteil VALUES (trunc(i+77600),259,'zweiundvierzig');
INSERT INTO Bauteil VALUES (trunc(i+77300),258,'dreiundvierzig');
INSERT INTO Bauteil VALUES (trunc(i+77000),257,'vierundvierzig');
INSERT INTO Bauteil VALUES (trunc(i+76700),256,'fünfundvierzig');
INSERT INTO Bauteil VALUES (trunc(i+76400),255,'sechsundvierzig');
INSERT INTO Bauteil VALUES (trunc(i+76100),254,'siebenundvierzig');
INSERT INTO Bauteil VALUES (trunc(i+75800),253,'achtundvierzig');
INSERT INTO Bauteil VALUES (trunc(i+75500),252,'neunundvierzig');
INSERT INTO Bauteil VALUES (trunc(i+75200),251,'fünfzig');
INSERT INTO Bauteil VALUES (trunc(i+74900),250,'einundfünfzig');
INSERT INTO Bauteil VALUES (trunc(i+74600),249,'zweiundfünfzig');
INSERT INTO Bauteil VALUES (trunc(i+74300),248,'dreiundfünfzig');
INSERT INTO Bauteil VALUES (trunc(i+74000),247,'vierundfünfzig');
INSERT INTO Bauteil VALUES (trunc(i+73700),246,'fünfundfünfzig');
INSERT INTO Bauteil VALUES (trunc(i+73400),245,'sechsundfünfzig');
```

```
INSERT INTO Bauteil VALUES (trunc(i+73100),244,'siebenundfünfzig');
INSERT INTO Bauteil VALUES (trunc(i+72800),243,'achtundfünfzig');
INSERT INTO Bauteil VALUES (trunc(i+72500),242,'neunundfünfzig');
INSERT INTO Bauteil VALUES (trunc(i+72200),241,'sechzig');
INSERT INTO Bauteil VALUES (trunc(i+71900),240,'einundsechzig');
INSERT INTO Bauteil VALUES (trunc(i+71600),239,'zweiundsechzig');
INSERT INTO Bauteil VALUES (trunc(i+71300),238,'dreiundsechzig');
INSERT INTO Bauteil VALUES (trunc(i+71000),237,'vierundsechzig');
INSERT INTO Bauteil VALUES (trunc(i+70700),236,'fünfundsechzig');
INSERT INTO Bauteil VALUES (trunc(i+70400),235,'sechsunsechzig');
INSERT INTO Bauteil VALUES (trunc(i+70100),234,'siebenundsechzig');
INSERT INTO Bauteil VALUES (trunc(i+69800),233,'achtundsechzig');
INSERT INTO Bauteil VALUES (trunc(i+69500),232,'neunundsechzig');
INSERT INTO Bauteil VALUES (trunc(i+69200),231,'siebzig');
INSERT INTO Bauteil VALUES (trunc(i+68900),230,'einundsiebzig');
INSERT INTO Bauteil VALUES (trunc(i+68600),229,'zweiundsiebzig');
INSERT INTO Bauteil VALUES (trunc(i+68300),228,'dreiundsiebzig');
INSERT INTO Bauteil VALUES (trunc(i+68000),227,'vierundsiebzig');
INSERT INTO Bauteil VALUES (trunc(i+67700),226,'fünfundsiebzig');
INSERT INTO Bauteil VALUES (trunc(i+67400),225,'sechsunsiebzig');
INSERT INTO Bauteil VALUES (trunc(i+67100),224,'siebenundsiebzig');
INSERT INTO Bauteil VALUES (trunc(i+66800),223,'achtundsiebzig');
INSERT INTO Bauteil VALUES (trunc(i+66500),222,'neunundsiebzig');
INSERT INTO Bauteil VALUES (trunc(i+66200),221,'achzig');
INSERT INTO Bauteil VALUES (trunc(i+65900),220,'einundachzig');
INSERT INTO Bauteil VALUES (trunc(i+65600),219,'zweiundachzig');
INSERT INTO Bauteil VALUES (trunc(i+65300),218,'dreiundachzig');
INSERT INTO Bauteil VALUES (trunc(i+65000),217,'vierundachzig');
INSERT INTO Bauteil VALUES (trunc(i+64700),216,'fünfundachzig');
INSERT INTO Bauteil VALUES (trunc(i+64400),215,'sechsunachzig');
INSERT INTO Bauteil VALUES (trunc(i+64100),214,'siebenundachzig');
INSERT INTO Bauteil VALUES (trunc(i+63800),213,'achtundachzig');
INSERT INTO Bauteil VALUES (trunc(i+63500),212,'neunundachzig');
INSERT INTO Bauteil VALUES (trunc(i+63200),211,'neunzig');
INSERT INTO Bauteil VALUES (trunc(i+62900),210,'einundneunzig');
INSERT INTO Bauteil VALUES (trunc(i+62600),209,'zweiundneunzig');
INSERT INTO Bauteil VALUES (trunc(i+62300),208,'dreiundneunzig');
INSERT INTO Bauteil VALUES (trunc(i+62000),207,'vierundneunzig');
INSERT INTO Bauteil VALUES (trunc(i+61700),206,'fünfundneunzig');
INSERT INTO Bauteil VALUES (trunc(i+61300),205,'sechsunneunzig');
INSERT INTO Bauteil VALUES (trunc(i+61000),204,'siebenundneunzig');
INSERT INTO Bauteil VALUES (trunc(i+60700),203,'achtundneunzig');
INSERT INTO Bauteil VALUES (trunc(i+60400),202,'neunundneunzig');
INSERT INTO Bauteil VALUES (trunc(i+60100),201,'hundert');
INSERT INTO Bauteil VALUES (trunc(i+59800),200,'hundertundeins');
INSERT INTO Bauteil VALUES (trunc(i+59500),199,'A');
INSERT INTO Bauteil VALUES (trunc(i+59200),198,'B');
```

```
INSERT INTO Bauteil VALUES (trunc(i+58900),197,'C');
INSERT INTO Bauteil VALUES (trunc(i+58600),196,'D');
INSERT INTO Bauteil VALUES (trunc(i+58300),195,'E');
INSERT INTO Bauteil VALUES (trunc(i+58000),194,'F');
INSERT INTO Bauteil VALUES (trunc(i+57700),193,'G');
INSERT INTO Bauteil VALUES (trunc(i+57400),192,'H');
INSERT INTO Bauteil VALUES (trunc(i+57100),191,'I');
INSERT INTO Bauteil VALUES (trunc(i+56800),190,'J');
INSERT INTO Bauteil VALUES (trunc(i+56500),189,'K');
INSERT INTO Bauteil VALUES (trunc(i+56200),188,'L');
INSERT INTO Bauteil VALUES (trunc(i+55900),187,'M');
INSERT INTO Bauteil VALUES (trunc(i+55600),186,'N');
INSERT INTO Bauteil VALUES (trunc(i+55300),185,'O');
INSERT INTO Bauteil VALUES (trunc(i+55000),184,'P');
INSERT INTO Bauteil VALUES (trunc(i+54700),183,'Q');
INSERT INTO Bauteil VALUES (trunc(i+54400),182,'R');
INSERT INTO Bauteil VALUES (trunc(i+54100),181,'S');
INSERT INTO Bauteil VALUES (trunc(i+53800),180,'T');
INSERT INTO Bauteil VALUES (trunc(i+53500),179,'U');
INSERT INTO Bauteil VALUES (trunc(i+53200),178,'V');
INSERT INTO Bauteil VALUES (trunc(i+52900),177,'W');
INSERT INTO Bauteil VALUES (trunc(i+52600),176,'X');
INSERT INTO Bauteil VALUES (trunc(i+52300),175,'Y');
INSERT INTO Bauteil VALUES (trunc(i+52000),174,'Z');
INSERT INTO Bauteil VALUES (trunc(i+51700),173,'I');
INSERT INTO Bauteil VALUES (trunc(i+51400),172,'II');
INSERT INTO Bauteil VALUES (trunc(i+51100),171,'III');
INSERT INTO Bauteil VALUES (trunc(i+50800),170,'IV');
INSERT INTO Bauteil VALUES (trunc(i+50500),169,'V');
INSERT INTO Bauteil VALUES (trunc(i+50200),168,'VI');
INSERT INTO Bauteil VALUES (trunc(i+49900),167,'VII');
INSERT INTO Bauteil VALUES (trunc(i+49600),166,'VIII');
INSERT INTO Bauteil VALUES (trunc(i+49300),165,'IX');
INSERT INTO Bauteil VALUES (trunc(i+49000),164,'X');
INSERT INTO Bauteil VALUES (trunc(i+48700),163,'XI');
INSERT INTO Bauteil VALUES (trunc(i+48400),162,'XII');
INSERT INTO Bauteil VALUES (trunc(i+48100),161,'XIII');
INSERT INTO Bauteil VALUES (trunc(i+47800),160,'XIV');
INSERT INTO Bauteil VALUES (trunc(i+47500),159,'XV');
INSERT INTO Bauteil VALUES (trunc(i+47200),158,'XVI');
INSERT INTO Bauteil VALUES (trunc(i+46900),157,'XVII');
INSERT INTO Bauteil VALUES (trunc(i+46600),156,'XVIII');
INSERT INTO Bauteil VALUES (trunc(i+46300),155,'XIX');
INSERT INTO Bauteil VALUES (trunc(i+46000),154,'XX');
INSERT INTO Bauteil VALUES (trunc(i+45700),153,'XXI');
INSERT INTO Bauteil VALUES (trunc(i+45400),152,'XXII');
INSERT INTO Bauteil VALUES (trunc(i+45100),151,'XXIII');
```

```
INSERT INTO Bauteil VALUES (trunc(i+44800),150,'XXIV');
INSERT INTO Bauteil VALUES (trunc(i+44500),149,'XXV');
INSERT INTO Bauteil VALUES (trunc(i+44200),148,'XXVI');
INSERT INTO Bauteil VALUES (trunc(i+43900),147,'XXVII');
INSERT INTO Bauteil VALUES (trunc(i+43600),146,'XXVIII');
INSERT INTO Bauteil VALUES (trunc(i+43300),145,'XXIX');
INSERT INTO Bauteil VALUES (trunc(i+43000),144,'XXX');
INSERT INTO Bauteil VALUES (trunc(i+42700),143,'XXXI');
INSERT INTO Bauteil VALUES (trunc(i+42400),142,'XXXII');
INSERT INTO Bauteil VALUES (trunc(i+42100),141,'XXXIII');
INSERT INTO Bauteil VALUES (trunc(i+41800),140,'XXXIV');
INSERT INTO Bauteil VALUES (trunc(i+41500),139,'XXXV');
INSERT INTO Bauteil VALUES (trunc(i+41200),138,'XXXVI');
INSERT INTO Bauteil VALUES (trunc(i+40900),137,'XXXVII');
INSERT INTO Bauteil VALUES (trunc(i+40600),136,'XXXVIII');
INSERT INTO Bauteil VALUES (trunc(i+40300),135,'XXXIX');
INSERT INTO Bauteil VALUES (trunc(i+40000),134,'XL');
INSERT INTO Bauteil VALUES (trunc(i+39700),133,'XLI');
INSERT INTO Bauteil VALUES (trunc(i+39400),132,'XLII');
INSERT INTO Bauteil VALUES (trunc(i+39100),131,'XLIII');
INSERT INTO Bauteil VALUES (trunc(i+38800),130,'XLIV');
INSERT INTO Bauteil VALUES (trunc(i+38500),129,'XLV');
INSERT INTO Bauteil VALUES (trunc(i+38200),128,'XLVI');
INSERT INTO Bauteil VALUES (trunc(i+37900),127,'XLVII');
INSERT INTO Bauteil VALUES (trunc(i+37600),126,'XLVIII');
INSERT INTO Bauteil VALUES (trunc(i+37300),125,'XLIX');
INSERT INTO Bauteil VALUES (trunc(i+37000),124,'L');
INSERT INTO Bauteil VALUES (trunc(i+36700),123,'AB');
INSERT INTO Bauteil VALUES (trunc(i+36400),122,'BC');
INSERT INTO Bauteil VALUES (trunc(i+36100),121,'CD');
INSERT INTO Bauteil VALUES (trunc(i+35800),120,'DE');
INSERT INTO Bauteil VALUES (trunc(i+35500),119,'EF');
INSERT INTO Bauteil VALUES (trunc(i+35200),118,'FG');
INSERT INTO Bauteil VALUES (trunc(i+34900),117,'GH');
INSERT INTO Bauteil VALUES (trunc(i+34600),116,'HI');
INSERT INTO Bauteil VALUES (trunc(i+34300),115,'IJ');
INSERT INTO Bauteil VALUES (trunc(i+34000),114,'JK');
INSERT INTO Bauteil VALUES (trunc(i+33700),113,'KL');
INSERT INTO Bauteil VALUES (trunc(i+33400),112,'LM');
INSERT INTO Bauteil VALUES (trunc(i+33100),111,'MN');
INSERT INTO Bauteil VALUES (trunc(i+32800),110,'NO');
INSERT INTO Bauteil VALUES (trunc(i+32500),109,'OP');
INSERT INTO Bauteil VALUES (trunc(i+32200),108,'PQ');
INSERT INTO Bauteil VALUES (trunc(i+31900),107,'QR');
INSERT INTO Bauteil VALUES (trunc(i+31600),106,'RS');
INSERT INTO Bauteil VALUES (trunc(i+31300),105,'ST');
INSERT INTO Bauteil VALUES (trunc(i+31000),104,'TU');
```

```
INSERT INTO Bauteil VALUES (trunc(i+30700),103,'UV');
INSERT INTO Bauteil VALUES (trunc(i+30400),102,'VW');
INSERT INTO Bauteil VALUES (trunc(i+30100),101,'WX');
INSERT INTO Bauteil VALUES (trunc(i+29800),100,'WY');
INSERT INTO Bauteil VALUES (trunc(i+29500),99,'YZ');
INSERT INTO Bauteil VALUES (trunc(i+29200),98,'AZ');
INSERT INTO Bauteil VALUES (trunc(i+28900),97,'BY');
INSERT INTO Bauteil VALUES (trunc(i+28600),96,'CX');
INSERT INTO Bauteil VALUES (trunc(i+28300),95,'DW');
INSERT INTO Bauteil VALUES (trunc(i+28000),94,'EV');
INSERT INTO Bauteil VALUES (trunc(i+27700),93,'FU');
INSERT INTO Bauteil VALUES (trunc(i+27400),92,'GT');
INSERT INTO Bauteil VALUES (trunc(i+27100),91,'HS');
INSERT INTO Bauteil VALUES (trunc(i+26800),90,'IR');
INSERT INTO Bauteil VALUES (trunc(i+26500),89,'JQ');
INSERT INTO Bauteil VALUES (trunc(i+26200),88,'KP');
INSERT INTO Bauteil VALUES (trunc(i+25900),87,'LO');
INSERT INTO Bauteil VALUES (trunc(i+25600),86,'MN');
INSERT INTO Bauteil VALUES (trunc(i+25300),85,'NM');
INSERT INTO Bauteil VALUES (trunc(i+25000),84,'OL');
INSERT INTO Bauteil VALUES (trunc(i+24700),83,'PK');
INSERT INTO Bauteil VALUES (trunc(i+24400),82,'QJ');
INSERT INTO Bauteil VALUES (trunc(i+24100),81,'RI');
INSERT INTO Bauteil VALUES (trunc(i+23800),80,'SH');
INSERT INTO Bauteil VALUES (trunc(i+23500),79,'TG');
INSERT INTO Bauteil VALUES (trunc(i+23200),78,'UF');
INSERT INTO Bauteil VALUES (trunc(i+22900),77,'VE');
INSERT INTO Bauteil VALUES (trunc(i+22600),76,'WD');
INSERT INTO Bauteil VALUES (trunc(i+22300),75,'XC');
INSERT INTO Bauteil VALUES (trunc(i+22000),74,'YB');
INSERT INTO Bauteil VALUES (trunc(i+21700),73,'ZA');
INSERT INTO Bauteil VALUES (trunc(i+21400),72,'AA');
INSERT INTO Bauteil VALUES (trunc(i+21100),71,'BB');
INSERT INTO Bauteil VALUES (trunc(i+20800),70,'CC');
INSERT INTO Bauteil VALUES (trunc(i+20500),69,'DD');
INSERT INTO Bauteil VALUES (trunc(i+20200),68,'EE');
INSERT INTO Bauteil VALUES (trunc(i+19900),67,'FF');
INSERT INTO Bauteil VALUES (trunc(i+19600),66,'GG');
INSERT INTO Bauteil VALUES (trunc(i+19300),65,'HH');
INSERT INTO Bauteil VALUES (trunc(i+19000),64,'II');
INSERT INTO Bauteil VALUES (trunc(i+18700),63,'JJ');
INSERT INTO Bauteil VALUES (trunc(i+18400),62,'KK');
INSERT INTO Bauteil VALUES (trunc(i+18100),61,'LL');
INSERT INTO Bauteil VALUES (trunc(i+17800),60,'MM');
INSERT INTO Bauteil VALUES (trunc(i+17500),59,'NN');
INSERT INTO Bauteil VALUES (trunc(i+17200),58,'OO');
INSERT INTO Bauteil VALUES (trunc(i+16900),57,'PP');
```

```
INSERT INTO Bauteil VALUES (trunc(i+16600),56,'QQ');
INSERT INTO Bauteil VALUES (trunc(i+16300),55,'RR');
INSERT INTO Bauteil VALUES (trunc(i+16000),54,'SS');
INSERT INTO Bauteil VALUES (trunc(i+15700),53,'TT');
INSERT INTO Bauteil VALUES (trunc(i+15400),52,'UU');
INSERT INTO Bauteil VALUES (trunc(i+15100),51,'VV');
INSERT INTO Bauteil VALUES (trunc(i+14800),50,'WW');
INSERT INTO Bauteil VALUES (trunc(i+14500),49,'XX');
INSERT INTO Bauteil VALUES (trunc(i+14200),48,'YY');
INSERT INTO Bauteil VALUES (trunc(i+13900),47,'ZZ');
INSERT INTO Bauteil VALUES (trunc(i+13600),46,'ABC');
INSERT INTO Bauteil VALUES (trunc(i+13300),45,'BCD');
INSERT INTO Bauteil VALUES (trunc(i+13000),44,'CDE');
INSERT INTO Bauteil VALUES (trunc(i+12700),43,'DEF');
INSERT INTO Bauteil VALUES (trunc(i+12400),42,'EFG');
INSERT INTO Bauteil VALUES (trunc(i+12100),41,'FGH');
INSERT INTO Bauteil VALUES (trunc(i+11800),40,'GHI');
INSERT INTO Bauteil VALUES (trunc(i+11500),39,'HIJ');
INSERT INTO Bauteil VALUES (trunc(i+11200),38,'IJK');
INSERT INTO Bauteil VALUES (trunc(i+10900),37,'JKL');
INSERT INTO Bauteil VALUES (trunc(i+10600),36,'KLM');
INSERT INTO Bauteil VALUES (trunc(i+10300),35,'LMN');
INSERT INTO Bauteil VALUES (trunc(i+10000),34,'MNO');
INSERT INTO Bauteil VALUES (trunc(i+9700),33,'NOP');
INSERT INTO Bauteil VALUES (trunc(i+9400),32,'OPQ');
INSERT INTO Bauteil VALUES (trunc(i+9100),31,'PQR');
INSERT INTO Bauteil VALUES (trunc(i+8800),30,'QRS');
INSERT INTO Bauteil VALUES (trunc(i+8500),29,'RST');
INSERT INTO Bauteil VALUES (trunc(i+8200),28,'STU');
INSERT INTO Bauteil VALUES (trunc(i+7900),27,'TUV');
INSERT INTO Bauteil VALUES (trunc(i+7600),26,'UVW');
INSERT INTO Bauteil VALUES (trunc(i+7300),25,'VWX');
INSERT INTO Bauteil VALUES (trunc(i+7000),24,'WXY');
INSERT INTO Bauteil VALUES (trunc(i+6700),23,'XYZ');
INSERT INTO Bauteil VALUES (trunc(i+6400),22,'ABCD');
INSERT INTO Bauteil VALUES (trunc(i+6100),21,'EFGH');
INSERT INTO Bauteil VALUES (trunc(i+5800),20,'IJKL');
INSERT INTO Bauteil VALUES (trunc(i+5500),19,'MNOP');
INSERT INTO Bauteil VALUES (trunc(i+5200),18,'QRST');
INSERT INTO Bauteil VALUES (trunc(i+4900),17,'UVWX');
INSERT INTO Bauteil VALUES (trunc(i+4600),16,'YZII');
INSERT INTO Bauteil VALUES (trunc(i+4500),15,'TEST');
INSERT INTO Bauteil VALUES (trunc(i+4200),14,'PHASE');
INSERT INTO Bauteil VALUES (trunc(i+3900),13,'UMFANG');
INSERT INTO Bauteil VALUES (trunc(i+3600),12,'BEWEIS');
INSERT INTO Bauteil VALUES (trunc(i+3300),11,'HOFFEN');
INSERT INTO Bauteil VALUES (trunc(i+3000),10,'DIPLOM');
```

```
INSERT INTO Bauteil VALUES (trunc(i+2700),9,'ARBEIT');
INSERT INTO Bauteil VALUES (trunc(i+2400),8,'NOTE');
INSERT INTO Bauteil VALUES (trunc(i+2100),7,'VORTRAG');
INSERT INTO Bauteil VALUES (trunc(i+1800),6,'ABSCHLUSS');
INSERT INTO Bauteil VALUES (trunc(i+1500),5,'VERTRAG');
INSERT INTO Bauteil VALUES (trunc(i+1200),4,'SEMINAR');
INSERT INTO Bauteil VALUES (trunc(i+900),3,'PAMT');
INSERT INTO Bauteil VALUES (trunc(i+600),2,'PROF');
INSERT INTO Bauteil VALUES (trunc(i+300),1,'DOKTOR');
INSERT INTO Bauteil VALUES (i,0,'ASSI');
END LOOP;
FOR j IN 1 .. 100 LOOP
INSERT INTO Lieferant VALUES (j,'AS','Berlin','Deutschland');
INSERT INTO Lieferant VALUES (trunc(j+100),'SA','Paris','Frankreich');
INSERT INTO Lieferant VALUES (trunc(j+200),'BQ','Moskau','Russland');
INSERT INTO Lieferant VALUES (trunc(j+300),'eG','Oslo','Norwegen');
INSERT INTO Lieferant VALUES (trunc(j+400),'AG','Stockholm','Schweden');
INSERT INTO Lieferant VALUES (trunc(j+500),'OHg','Helsinki','Finnland');
INSERT INTO Lieferant VALUES (trunc(j+600),'GmbH','Rom','Italien');
INSERT INTO Lieferant VALUES (trunc(j+700),'eV','Madrid','Spanien');
INSERT INTO Lieferant VALUES (trunc(j+800),'TEST','Rio','Brasilien');
INSERT INTO Lieferant VALUES (trunc(j+900),'TAT','Athen','Griechenland');
INSERT INTO Lieferant VALUES (trunc(j+1000),'UNI','New York','USA');
INSERT INTO Lieferant VALUES (trunc(j+1100),'FAK','Mexiko City','Mexiko');
INSERT INTO Lieferant VALUES (trunc(j+1200),'BOSS','Toronto','Kanada');
INSERT INTO Lieferant VALUES (trunc(j+1300),'CHEF','London','GB');
INSERT INTO Lieferant VALUES (trunc(j+1400),'KANN','Dublin','Irland');
INSERT INTO Lieferant VALUES (trunc(j+1500),'MUSS','Kapstadt','Südafrika');
INSERT INTO Lieferant VALUES (trunc(j+1600),'IST','Kairo','Ägypten');
INSERT INTO Lieferant VALUES (trunc(j+1700),'DORF','Minsk','Weißrußland');
INSERT INTO Lieferant VALUES (trunc(j+1800),'STADT','Prag','Tschechien');
INSERT INTO Lieferant VALUES (trunc(j+1900),'SUPER','Tokio','Japan');
INSERT INTO Lieferant VALUES (trunc(j+2000),'JESU','Peking','China');
INSERT INTO Lieferant VALUES (trunc(j+2100),'AS','AFG','Maryland');
INSERT INTO Lieferant VALUES (trunc(j+2200),'SA','FNUISA','Karlifonien');
INSERT INTO Lieferant VALUES (trunc(j+2300),'BQ','IOSD','Texas');
INSERT INTO Lieferant VALUES (trunc(j+2400),'eG','KAE','Florida');
INSERT INTO Lieferant VALUES (trunc(j+2500),'AG','SAAB','Washington');
INSERT INTO Lieferant VALUES (trunc(j+2600),'OHg','TIKI','Montana');
INSERT INTO Lieferant VALUES (trunc(j+2700),'GmbH','KUSS','Kansas');
INSERT INTO Lieferant VALUES (trunc(j+2800),'eV','WQR','Serbien');
INSERT INTO Lieferant VALUES (trunc(j+2900),'TEST','WERT','Mongolei');
INSERT INTO Lieferant VALUES (trunc(j+3000),'TAT','SONG','Tibet');
INSERT INTO Lieferant VALUES (trunc(j+3100),'UNI','GONG','Bhali');
INSERT INTO Lieferant VALUES (trunc(j+3200),'FAK','FLUG','Thailand');
INSERT INTO Lieferant VALUES (trunc(j+3300),'BOSS','TRUG','Indonesien');
INSERT INTO Lieferant VALUES (trunc(j+3400),'CHEF','AFK','Kongo');
```

```

INSERT INTO Lieferant VALUES (trunc(j+3500), 'KANN', 'WTF', 'Sudan');
INSERT INTO Lieferant VALUES (trunc(j+3600), 'MUSS', 'BRB', 'Kuwait');
INSERT INTO Lieferant VALUES (trunc(j+3700), 'IST', 'HV', 'Tschad');
INSERT INTO Lieferant VALUES (trunc(j+3800), 'DORF', 'MD', 'Namibia');
INSERT INTO Lieferant VALUES (trunc(j+3900), 'STADT', 'KINO', 'Somalia');
INSERT INTO Lieferant VALUES (trunc(j+4000), 'SUPER', 'FILM', 'Pakistan');
INSERT INTO Lieferant VALUES (trunc(j+4100), 'JESU', 'DIVX', 'Indien');
INSERT INTO Lieferant VALUES (trunc(j+4200), 'AS', 'MPG', 'Argentinien');
INSERT INTO Lieferant VALUES (trunc(j+4300), 'SA', 'OGM', 'Venezuela');
INSERT INTO Lieferant VALUES (trunc(j+4400), 'BQ', 'MP3', 'Kuba');
INSERT INTO Lieferant VALUES (trunc(j+4500), 'eG', 'WMA', 'Island');
INSERT INTO Lieferant VALUES (trunc(j+4600), 'AG', 'WMF', 'Türkei');
INSERT INTO Lieferant VALUES (trunc(j+4700), 'OHg', 'QTIME', 'Peru');
INSERT INTO Lieferant VALUES (trunc(j+4800), 'GmbH', 'QUAD', 'Kambodscha');
INSERT INTO Lieferant VALUES (trunc(j+4900), 'eV', 'ZUCKER', 'Australien');
END LOOP;
END;

/* extra ausführen! */

INSERT INTO LAGER (Lieferanten_ID, Bauteil_ID)
SELECT L.Lieferanten_ID, B.Bauteil_ID
FROM Lieferant L, Bauteil B
WHERE Lieferanten_ID BETWEEN '41' AND '561'
AND Bauteil_ID BETWEEN '55555' AND '61435';

/* = 2363630 Datensätze, bei vollem Umfang Fehler (Space nicht erweiterbar) */

/* index nach insert auf lager durchführen */

CREATE BITMAP INDEX Bauteil_Lieferant_Land
ON Lager (B.Bauteil_Typ, LI.Land)
FROM Lager LA, Bauteil B, Lieferant LI
WHERE LA.Bauteil_ID = B.Bauteil_ID
AND LA.Lieferanten_ID = LI.Lieferanten_ID;

/* nach commit und statistiken ausführen */

rem Anführungsplaene ansehen
@utlxplan

rem Performance-Messung
SET TIMING ON;

DELETE Plan_Table;

```

```
EXPLAIN PLAN FOR
SELECT /*+ INDEX(LAGER BAUTEIL_LIEFERANT_LAND) */ Name_Lieferant
FROM Bauteil NATURAL JOIN Lager NATURAL JOIN Lieferant
WHERE Bauteil_Typ = 'achtundachzig' AND Land = 'Deutschland';
```

```
select
    substr (lpad(' ', level-1) || operation || '
(' || options || ')',1,30 ) "Operation",
    object_name "Object"
from plan_table
start with id = 0
connect by prior id=parent_id;
```

```
SET TIMING OFF
```

```
DROP INDEX Bauteil_Lieferant_Land;
DROP TABLE Lieferant CASCADE CONSTRAINTS;
DROP TABLE Lager CASCADE CONSTRAINTS;
DROP TABLE Bauteil CASCADE CONSTRAINTS;
```

Literaturverzeichnis

- [ACK⁺04] Agrawal, S.; Chaudhuri, S.; Kollár, L.; Marathe, A. P.; Narasayya, V. R.; Syamala, M.: Database Tuning Advisor for Microsoft SQL Server 2005. In *VLDB '04*, S. 1110–1121, 2004.
- [BC06] Bruno, N.; Chaudhuri, S.: To Tune or not to Tune? A Lightweight Physical Design Alerter. In *VLDB '06*, S. 499–510, 2006.
- [BC07] Bruno, N.; Chaudhuri, S.: An online approach to physical design tuning. In *ICDE*, 2007. not published yet.
- [BM72] Bayer, R.; McCreight, E. M.: Organization and maintenance of large ordered indices. *Acta Inf.*, Band 1, S. 173–189, 1972.
- [Bur02] Burleson, D.: How to use oracle9i bitmap join indexes. <http://www.dba-oracle.com/art-builder-bitmap-join-idx.htm>, November 2002. Visited January - March 2007.
- [CCS93] Codd, E. F.; Codd, S. B.; Salley, C. T.: Providing olap (online analytical processing) to user-analysis: An it mandate. WhitePaper, E. F. Codd and Associates, 1993.
- [CDF⁺01] Chong, E. I.; Das, S.; Freiwald, C.; Srinivasan, J.; Yalamanchi, A.; Jagannath, M.; Tran, A.-T.; Krishnan, R.: B+-tree indexes with hybrid row identifiers in oracle8i. In *ICDE '01: Proceedings of the 17th International Conference on Data Engineering*, S. 341. IEEE Computer Society, Washington, DC, USA, 2001.
- [CFM95] Caprara, A.; Fischetti, M.; Maio, D.: Exact and Approximate Algorithms for the Index Selection Problem in Physical Database Design. *IEEE Transactions on Knowledge and Data Engineering*, Band 7, Nr. 6, S. 955–967, 1995.
- [CI98] Chan, C.-Y.; Ioannidis, Y. E.: Bitmap index design and evaluation. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, S. 355–366. ACM Press, New York, NY, USA, 1998.
- [Com78] Comer, D.: The Difficulty of Optimum Index Selection. *ACM Transactions on Database Systems*, Band 3, Nr. 4, S. 440–445, 1978.
- [Cor05] Corporation, I.: An architectural blueprint for autonomic computing. White Paper, June 2005. Third Edition.

- [DB06] Danchenkov, A. B.; Burleson, D. K.: *Oracle Tuning: The Definitive Reference*. Rampant Techpress, 2006.
- [HK83] Hommel, G.; Krönig, D. (Hrsg.): *Requirements Engineering, Arbeitstagung der GI, Friedrichshafen, 12.-14. Oktober 1983*, Informatik-Fachberichte, Band 74. Springer, 1983.
- [KPP04] Kellerer, H.; Pferschy, U.; Pisinger, D.: *Knapsack Problems*. Springer-Verlag, Berlin, Heidelberg, 2004.
- [Leh03] Lehner, W.: *Datenbanktechnologie für Data-Warehouse-Systeme: Konzepte und Methoden*, S. 55,83,85–92. dpunkt-Verlag/Heidelberg, 1. Auflage, 2003.
- [LS02] Lane, P.; Schupmann, V.: *Oracle9i Data Warehousing Guide, Release 2 (9.2)*, S. 122–125. Oracle Corporation, 2002.
- [LSS07] Lühring, M.; Sattler, K.; Schallehn, E.; Schmidt, K.: Autonomes Index Tuning – DBMS-integrierte Verwaltung von Soft Indexen. In *BTW Proceedings*, 2007. To appear.
- [Mei05] Meier, A.: Query optimierung - einföhrung ins tuning von datenbank-abfragen am beispiel von oracle. <http://www.ite.ethz.ch/kids/meier>, White Paper, Oktober 2005. Visited March 2007.
- [NS01] Nehmer, J.; Sturm, P.: *Systemsoftware: Grundlagen moderner Betriebssysteme*, S. 112–136. dpunkt-Verlag/Heidelberg, 2. Auflage, 2001.
- [SHS00] Saake, G.; Heuer, A.; Sattler, K.-U.: *Datenbanken: Konzepte und Sprachen*, S. 106–108. mitp-Verlag/Bonn, 2. Auflage, 2000.
- [SHS05] Saake, G.; Heuer, A.; Sattler, K.-U.: *Datenbanken: Implementierungstechniken*, S. 16,50,108,144–151,261–268,334,351–357. mitp-Verlag/Bonn, 2. Auflage, 2005.
- [SM06] Sun Microsystems, I.: Sun blade 1500 product notes. <http://docs.sun.com/source/817-5131-14/index.html>, 2006. Visited March 2007.
- [SPL03] Seacord, R. C.; Plakosh, D.; Lewis, G. A.: *Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices*. Addison-Wesley Professional, 2003.
- [SSG04] Sattler, K.; Schallehn, E.; Geist, I.: Autonomous Query-driven Index Tuning. In *Proc. Int. Database Engineering and Applications Symposium (IDEAS 2004)*, Coimbra, Portugal, S. 439–448, Juli 2004.
- [Val87] Valduriez, P.: Join indices. *ACM Trans. Database Syst.*, Band 12, Nr. 2, S. 218–246, 1987.
- [WHMZ94] Weikum, G.; Hasse, C.; Moenkeberg, A.; Zabback, P.: The COMFORT Automatic Tuning Project, Invited Project Review. *Information Systems*, Band 19, Nr. 5, S. 381–432, 1994.

-
-
- [ZRL⁺04] Zilio, D. C.; Rao, J.; Lightstone, S.; Lohman, G. M.; Storm, A.; Garcia-Arellano, C.; Fadden, S.: DB2 Design Advisor: Integrated Automatic Physical Database Design. In *VLDB '04*, S. 1087–1097, 2004.

Selbstständigkeitserklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbstständig und nur mit erlaubten Hilfsmitteln angefertigt habe.

Magdeburg, den 12. April 2007

Vorname Name

