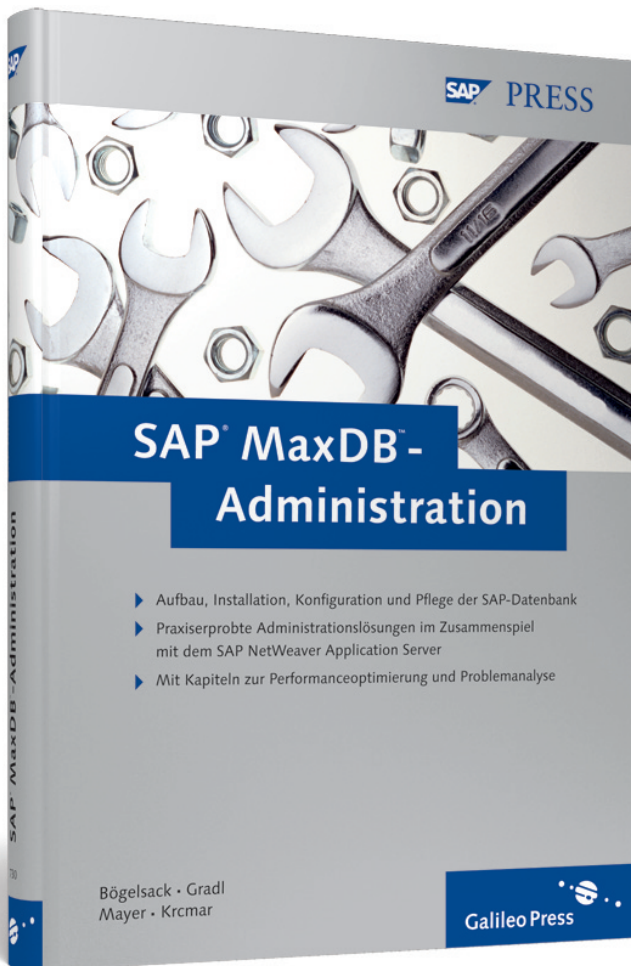


André Bögelsack, Stephan Gradl,
Manuel Mayer, Helmut Krcmar

SAP® MaxDB™-Administration




Galileo Press

Bonn • Boston

Auf einen Blick

1	Einführung in SAP MaxDB	11
2	Überblick SAP MaxDB	23
3	SAP MaxDB und SAP	99
4	Administrationsaufgaben	127
5	Performance-Tuning	231
6	Problemsituationen	313
7	Zusammenfassung und Ausblick	339
A	Befehlsreferenz dbmcli	341
B	Die Autoren	353
	Index	355

Inhalt

1	Einführung in SAP MaxDB	11
1.1	Entstehungsgeschichte	11
1.2	Features der SAP MaxDB	12
1.2.1	Generelle Features	13
1.2.2	Flexibilität beim Betrieb	14
1.2.3	SQL-Modi und Schnittstellen	15
1.2.4	Einsatzgebiete	16
1.3	Nützliche Internetquellen	17
1.3.1	Offizielle SAP MaxDB-Webseite	17
1.3.2	SAP MaxDB-Wiki im SAP Developer Network	18
1.3.3	SAP MaxDB-FAQ	18
1.3.4	SAP MaxDB-Forum	18
1.4	Aufbau dieses Buches	18
2	Überblick SAP MaxDB	23
2.1	SAP MaxDB-Instanztypen	23
2.1.1	OLTP und OLAP	23
2.1.2	SAP liveCache	25
2.2	SAP MaxDB-Software	29
2.2.1	X-Server	29
2.2.2	Database Studio	30
2.2.3	Database Manager GUI	33
2.2.4	Database Manager CLI	37
2.2.5	SQL Studio	39
2.2.6	SQL CLI	40
2.2.7	Web SQL	41
2.2.8	Weitere Hilfsmittel	43
2.3	SAP MaxDB-Benutzerkonzept	50
2.3.1	MaxDB-Benutzer	51
2.3.2	Betriebssystembenutzer	58
2.3.3	Sicherheitsaspekte	60
2.4	Datenbankkonzepte	61
2.4.1	Kernel Threads	62
2.4.2	Caches	71
2.4.3	Daten- und Log Volumes	79

2.4.4	Savepoints und Snapshots	86
2.4.5	Sperren	88
2.4.6	Verzeichnisstruktur	90
2.4.7	Betriebszustände	93
2.4.8	Datenbankparameter	94
2.4.9	Konfigurationsdateien	96
2.5	Zusammenfassung	97
3	SAP MaxDB und SAP	99
3.1	SAP-Architekturen	99
3.1.1	ABAP- und Java-Stack	100
3.1.2	Architekturebenen	101
3.2	Kommunikation mit der SAP MaxDB	104
3.2.1	SAP MaxDB Interfaces	107
3.2.2	Kommunikation mit SAP-Systemen	110
3.3	Wichtige Transaktionen	114
3.3.1	DB50 – Datenbankassistent	114
3.3.2	DB13	121
3.3.3	RZ20	124
3.4	Zusammenfassung	126
4	Administrationsaufgaben	127
4.1	Server-Software-Installation und Upgrade	127
4.1.1	SDBINST/SDBSETUP	128
4.1.2	SDBUPD	139
4.2	Datenbank erzeugen und initialisieren	143
4.2.1	Datenbank planen	144
4.2.2	Datenbank per GUI erzeugen	146
4.2.3	Datenbank per dbmcli erzeugen	155
4.2.4	Zusammenspiel mit SAPInst	160
4.3	Datenbank konfigurieren	162
4.3.1	Data/Log Volumes hinzufügen und löschen	162
4.3.2	Log Volumes und Log-Modus konfigurieren	170
4.3.3	Update der Systemtabellen	175
4.3.4	Parameteränderungen	176
4.4	Datenbank-Backup	180
4.4.1	Konzepte der Sicherung	181
4.4.2	Backup-Medium anlegen	184

4.4.3	Inkrementelle und vollständige Sicherung	189
4.4.4	Log-Sicherungen	192
4.4.5	Snapshots	197
4.4.6	Überprüfung von Sicherungen	202
4.5	Datenbank-Recovery	206
4.5.1	Recovery-Typen	206
4.5.2	Recovery-Strategie	207
4.5.3	Recovery/Recovery mit Initialisierung	209
4.5.4	Fehlerhafte Log Mirrors reintegrieren	213
4.5.5	Bad Indexes	215
4.6	Konsistenzprüfungen	218
4.6.1	Allgemeine Erklärung	218
4.6.2	Datenbankstruktur überprüfen	219
4.7	Datenbank löschen	222
4.7.1	Datenbank löschen	222
4.7.2	Server-Software-Deinstallation	226
4.8	Zusammenfassung	229

5 Performance-Tuning 231

5.1	Performanceoptimierung	232
5.2	Indizes	233
5.2.1	B*-Bäume: Theorie	233
5.2.2	Primär- und Sekundärschlüssel	235
5.3	Der Datenbankoptimierer	243
5.3.1	Grundlagen	243
5.3.2	Kriterien für die Auswahl bestimmter Zugriffsstrategien	249
5.4	Caches	252
5.4.1	Motivation	253
5.4.2	Die verschiedenen Caches	253
5.4.3	Die richtige Größe der Caches	257
5.4.4	Die wichtigsten Informationen über Caches	257
5.4.5	Statistiken der kritischen Regionen	263
5.5	Analyse-Tools	265
5.5.1	Database Analyzer	266
5.5.2	Resource Monitor	278
5.5.3	Command Monitor	282
5.5.4	SQL-Explain	289
5.6	Performance mit SAP NetWeaver AS	293
5.6.1	Performance-Analyse SAP NetWeaver AS	293

5.6.2	Lastanalyse	296
5.6.3	Datenbankanalyse im SAP NetWeaver AS	297
5.7	Zusammenfassung	312

6 Problemsituationen 313

6.1	Diagnosedateien	313
6.1.1	Dev-Traces	314
6.1.2	SQL-Trace	314
6.1.3	SQLDBC-Trace	315
6.1.4	X-Server-Protokoll: xserver_<hostname>.prt	316
6.1.5	appldiag	317
6.1.6	dbm.prt	317
6.1.7	KnIMsg (knldiag)	318
6.1.8	KnIMsgArchive (knldiag.err, dbm.utl)	319
6.1.9	dbm.knl	320
6.1.10	dbm.ebp	321
6.1.11	dbm.ebl	321
6.1.12	rtedump	322
6.1.13	knltrace	323
6.1.14	knldump	324
6.2	Fehlerarten und Analyse	325
6.2.1	Installationsprobleme	325
6.2.2	Verbindungsprobleme	326
6.2.3	Log Full/Data Full	327
6.2.4	Systemabsturz/Systemfehler	331
6.2.5	Systemblockade	332
6.2.6	Sicherungs-/Wiederherstellungsfehler	335
6.2.7	Hardwarefehler	336
6.3	Zusammenfassung	337

7 Zusammenfassung und Ausblick 339

Anhang 341

A	Befehlsreferenz dbmcli	341
B	Die Autoren	353
	Index	355

Caches, Indizes, Analysewerkzeuge und wie sie am besten verwendet werden – in diesem Kapitel machen wir Sie mit den Hintergründen vertraut und zeigen Ihnen, wie Sie die Ursachen von Performance-Engpässen erkennen und beheben.

5 Performance-Tuning

Datenbanken stellen sowohl die Persistenz als auch die Integrität der Daten sicher. Dass Datenbanken jedoch eine so breite Verwendung auf fast sämtlichen Gebieten der EDV finden, liegt zu einem großen Teil an den sehr schnellen und flexiblen Zugriffsmöglichkeiten auf die gespeicherten Informationen. In diesem Kapitel werden wir zunächst die theoretischen und technischen Grundlagen aufzeigen, die diese performanten Zugriffe ermöglichen. Außerdem werden wir Mittel und Methoden vorstellen, wie Sie Leistungsengpässe erkennen, analysieren und beheben können.

Dazu erläutern wir in Abschnitt 5.1 zuerst den Begriff *Performance* und grenzen die Möglichkeiten des Datenbankadministrators hinsichtlich der Performanceoptimierung ein. In Abschnitt 5.2 stellen wir den Aufbau des Datenspeicherungskonzepts, den theoretischen Hintergrund der in der SAP MaxDB verwendeten Suchstruktur, den B*-Baum, sowie dessen Ausprägungen für Primär- und Sekundärindizes dar. Wann und auf welche Weise diese Suchstrukturen beim Zugriff auf die Daten verwendet werden, und wie Sie die nötigen Informationen für einen optimalen Zugriff bereitstellen können, erklären wir in Abschnitt 5.3. Außerdem erläutern wir, wie die Ausführung langsamer SQL-Statements beschleunigt werden kann. Warum Zugriffe trotz dieser Suchstrukturen im Gegensatz zum Lesen der Daten aus dem Hauptspeicher um Größenordnungen langsamer sind, und wie dieser Geschwindigkeitsvorteil des Hauptspeichers in der MaxDB ausgenutzt wird, führen wir im Abschnitt über Caches (Abschnitt 5.4) aus. Im Abschnitt 5.5 zeigen wir Ihnen, wie Sie die Datenbank mit Hilfe des DB Analyzers überwachen können. Außerdem veranschaulichen wir, wie man mit dem Resource Monitor die SQL-

Statements in Erfahrung bringt, welche die größte Last auf der Datenbank verursachen, wie Sie mit dem Command Monitor einzelne teure SQL-Statements aufspüren, und wie Sie diese schließlich mit der SQL-Explain-Anweisung analysieren können. Der Analysevorgang mit Hilfe des SAP NetWeaver AS ist Gegenstand des letzten Abschnitts. Hier stellen wir dar, wie Performanceengpässe mit den Transaktionen eines SAP-Systems erkannt und die Ursachen analysiert und beseitigt werden können.

5.1 Performanceoptimierung

Performance Ein zentraler Aspekt der Datenbankperformance ist die Schnelligkeit des Abarbeitens von SQL-Statements. Je schneller Anfragen bearbeitet werden, desto performanter ist das Datenbanksystem. Sie können die Performance einer Datenbank also beeinflussen, indem Sie dafür sorgen, dass die Datenbank die zu erwartenden Abfragen ideal unterstützt. Auf diese Weise verursachen die Statements weniger Kosten, werden also »billiger«.

Abfragen sind teuer, ...

- ▶ wenn sie eine große Menge von Daten abfragen, mit einem eventuell hohen Anteil an überflüssigen Daten,
- ▶ wenn für die Ausführung eine oder mehrere Tabellen vollständig durchsucht werden müssen.

Der erste dieser beiden Fälle liegt vollständig in der Hand des Datenbankprogrammierers. Diese Art Anfragen kann nur durch eine variable physikalische Clusterung der Daten im Hintergrundspeicher beschleunigt werden, was jedoch von vielen Datenbanksystemen für sekundäre Indizes nicht unterstützt wird – auch nicht von MaxDB.

Optimierung Den zweiten Typ von kostenintensiven Anfragen hingegen können wir durch geeignetes »Tuning« der Datenbank optimieren und damit günstiger machen. Um den eben angesprochenen Typ von Anfragen zu optimieren, ist es zunächst notwendig zu verstehen, wie die Anfragen im Datenbanksystem generell ausgeführt werden. Dies geschieht mithilfe eines *Ausführungsplans*. Ein Ausführungsplan wird bei jeder Anfrage neu erzeugt und legt die Art des Zugriffes auf die Daten fest. Eine intuitive Möglichkeit ist, die gesamte Tabelle zu durchsuchen. Eine weitere Möglichkeit ist der Gebrauch von Daten-

strukturen, welche die Suche vor allem bei großen Tabellen extrem beschleunigen können. Die betreffenden Datenstrukturen und deren Verwendung werden wir unter anderem im folgenden Abschnitt darstellen.

5.2 Indizes

Bei der heutigen Größe von Datenbanken, die in einzelnen Fällen mehrere Petabyte an Umfang erreichen, ist es nötig, diese Daten auf Festplatten zu speichern, da sie nicht im Hauptspeicher vorgehalten werden können. Da jedoch die Zugriffe auf Festplatten um Größenordnungen länger dauern als Zugriffe auf den Hauptspeicher, ist die Häufigkeit der für das Auslesen eines Datums nötigen Plattenzugriffe für die Performance ein ausschlaggebender Faktor. Aus diesem Grund wurden auch die in Abschnitt 5.3, »Der Datenbankoptimierer«, beschriebenen SQL Optimizer in allen Datenbanksystemen implementiert, um zu entscheiden welche Zugriffsstrategie die wenigsten Plattenzugriffe erfordert und somit am performantesten ist. Im Folgenden erläutern wir zuerst die theoretischen Konzepte, die es ermöglichen, einen Datensatz auf einem Hintergrundspeicher mit einer garantierten maximalen Anzahl von Plattenzugriffen auszulesen. Nach der Behandlung der theoretischen Grundlagen stellen wir die Eigenschaften von B*-Bäumen in MaxDB dar.

5.2.1 B*-Bäume: Theorie

Wie bereits der Name sagt, handelt es sich bei dieser Datenstruktur um einen Baum. Ein Baum ist eine robuste und leistungsfähige Datenstruktur und in nahezu jedem modernen Datenbanksystem und auch bereits in den führenden Dateisystemen integriert. Häufig wird sie als die Datenstruktur des relationalen Datenbankmodells bezeichnet.

B*-Baum

Der B*-Baum benutzt in den inneren Knoten lediglich Referenzschlüssel, die nicht zwangsläufig realen Schlüssel entsprechen müssen. Bei der MaxDB ist dies zwar der Fall, jedoch ist es für diese Datenstruktur nicht zwingend notwendig und der Implementierung des Datenbankherstellers überlassen. Da jeder Knoten eine komplette Seite des Hintergrundspeichers belegt, können viele dieser Referenzschlüssel in einem Knoten gespeichert werden. Dadurch entstehen

Aufbau

auch bei großen Datenmengen wenige Ebenen im Baum, was zu weniger Plattenzugriffen für das Auffinden eines Datensatzes führt. Erst in der untersten Ebene, bei den Blättern, erfolgt die Zuordnung von reellen Schlüssel zu den Daten. Auf dieser Ebene wird eine weitere Optimierung der Datenstruktur für das sequenzielle Auslesen implementiert. Jede Seite des Hintergrundspeichers enthält zusätzlich Verweise auf die vorhergehende und die nachfolgende Seite. Ist also der Einstiegspunkt einmal gefunden, so muss nur noch so lange den sequenziellen Referenzen gefolgt werden, bis das Suchprädikat nicht mehr erfüllt ist (siehe Abbildung 5.1).

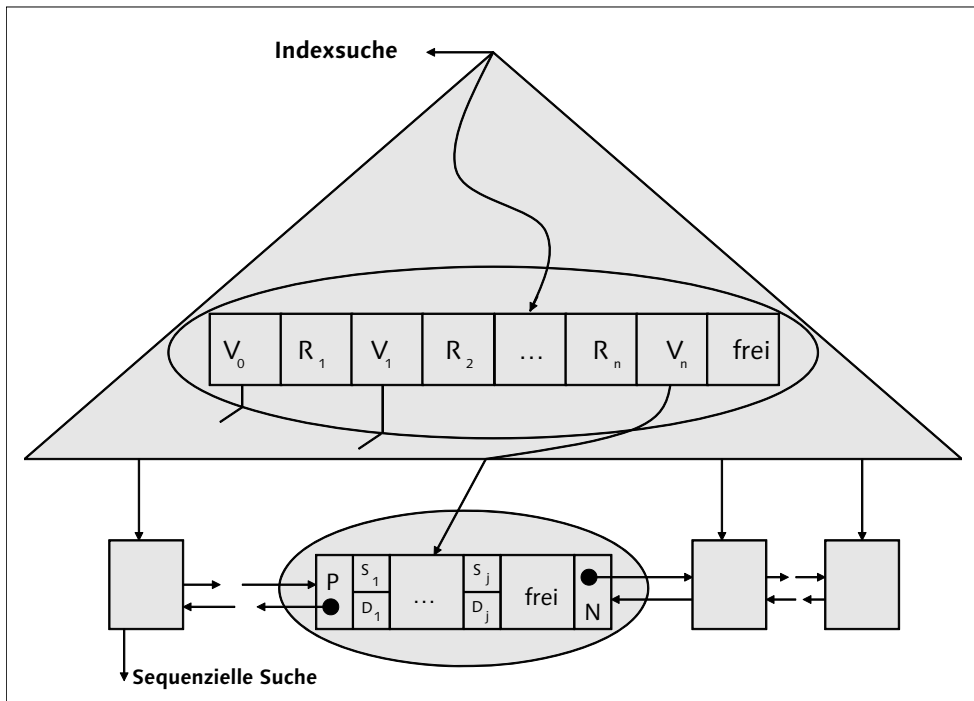


Abbildung 5.1 Schematischer Aufbau eines B*-Baums

Ausgleichen der Werteverteilung

Die Algorithmen für das Einfügen und das Löschen von Daten sind so gestaltet, dass der Baum immer *balanciert* bleibt. Dies bedeutet, dass der Weg von der Wurzel des Baumes bis zu jedem Blatt, also zu einem Datensatz, immer gleich lang ist.

Die großen Vorteile des B*-Baums möchten wir Ihnen durch den Vergleich mit dem B-Baum verdeutlichen. Da es sich bei dieser Daten-

struktur um einen internen Suchbaum handelt, der die Daten auch in den Knoten abspeichert, ist er nicht so gut an die Eigenschaften von Hintergrundspeichern angepasst wie der B*-Baum, der eine noch größere Anzahl an Datenseiten bei gleicher Höhe des Baumes referenziert. Je niedriger der Baum, desto weniger Zugriffe sind nötig, um einen Datensatz zu erreichen

Hat ein Baum 4 Ebenen, und finden auf jedem inneren Knoten 200 Referenzschlüssel Platz, so werden mindestens $1,6 \times 10^7$ Elemente, also Datensätze, referenziert. Bei gleicher Höhe, also bei der gleichen maximalen Anzahl von Plattenzugriffen, die für das Auffinden eines Datensatzes nötig sind, kann dieser Baum auf eine Größe von $2,56 \times 10^{10}$ Elemente wachsen, ohne an Performanz einzubüßen. Da ein Teil eines Indexes im Cache ist, sind häufig nur 2–3 Plattenzugriffe nötig, um einen Datensatz aus einer Menge von 10 Milliarden Datensätzen zu finden. Bei 1 KB pro Datensatz entspricht dies einer Tabellengröße von ca. 1 TB.

Zahlenbeispiel

Nachdem wir Ihnen die theoretischen Eigenschaften des B*-Baums erläutert haben, veranschaulichen wir Ihnen im nächsten Abschnitt, an welchen Stellen B*-Bäume integriert sind und wie sie verwendet werden.

5.2.2 Primär- und Sekundärschlüssel

MaxDB verwendet B*-Bäume, die direkt in den Tabellen mit gespeichert sind. Abbildung 5.2 zeigt den Aufbau eines B*-Baums in der MaxDB. Dieser wird von »unten«, also aus Richtung der Blattebene, erstellt. Auf dieser untersten Ebene sind die Datensätze in aufsteigender Folge bezüglich des Primärschlüssels angeordnet.

Ausprägung des B*-Baums

Die Knoten der Indexebene werden aus den Werten der Blattebene aufgebaut. Ist das Ende einer Seite auf der Blattebene erreicht, so wird auf der Indexebene ein neuer Eintrag erstellt, der eine signifikante Unterscheidung des letzten Eintrags der Seite zur nachfolgenden darstellt. Im Beispiel ist dies der Fall bei Hamburg. In der Liste der Städte wäre Halle der letzte Eintrag auf der vorhergehenden Seite. Deshalb muss der Eintrag in der Indexebene »HAM« heißen. Das Erstellen des Primärindex funktioniert nun so weiter bis alle Verweise auf einer Seite, der *Root Page*, Platz finden.

Beispielhafter Aufbau

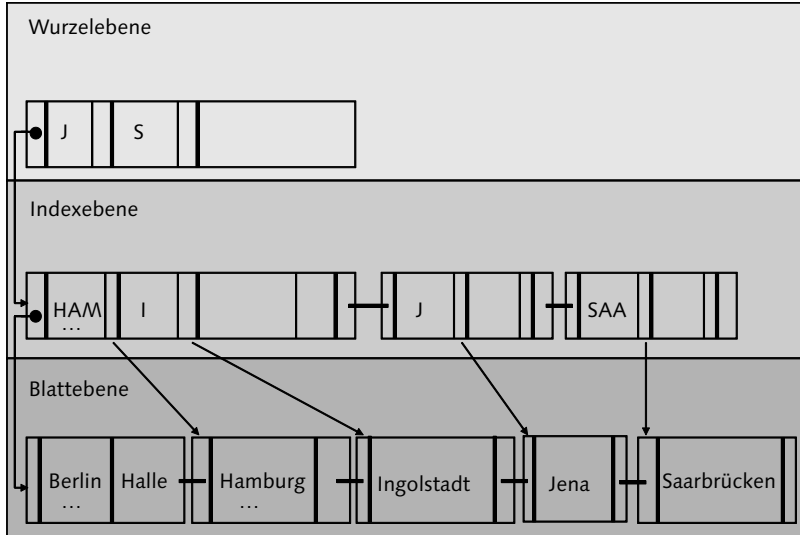


Abbildung 5.2 Beispielhafte Speicherung einer Tabelle in einem B*-Baum

Hinzufügen eines Datensatzes

Werden nun im Verlaufe der Benutzung der Tabelle weitere Datensätze eingefügt, sodass auf der Wurzelebene nicht mehr alle Verweise Platz finden, so wird diese Wurzelebene geteilt und die beiden resultierenden Seiten werden zu Indexseiten, auf welche nun eine neue Wurzelebene verweist. Beispielhaft wird dies in Abbildung 5.3 und Abbildung 5.4 dargestellt.

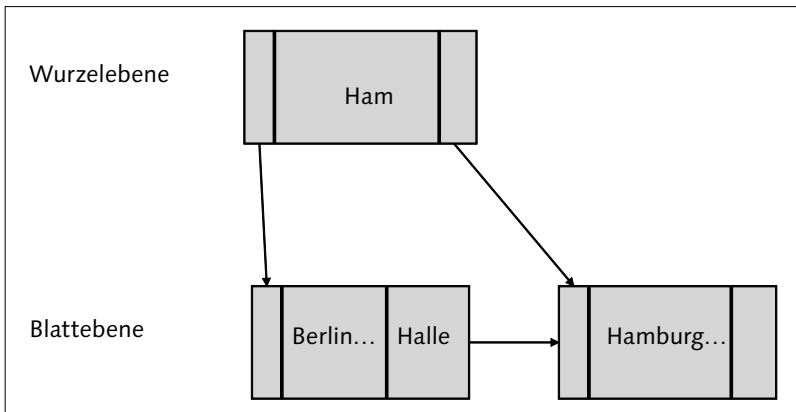


Abbildung 5.3 Situation vor dem Einfügen

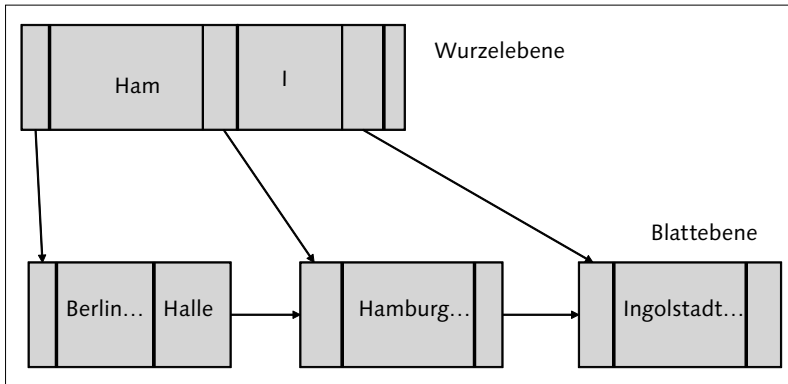


Abbildung 5.4 Situation nach dem Einfügen

Dabei sind die Einträge auf allen Ebenen mit sequenziellen Links verbunden, sodass auch Range Querys performant ausgeführt werden können. Die maximale Anzahl von Tabelleneinträgen ist beschränkt, insofern der B*-Baum des Primärexes in der MaxDB auf eine Höhe von vier Indexebenen und einer Wurzelebene beschränkt ist. Da jedoch eine logische Seite 8 KB groß ist, können ausreichend große Tabellen verwaltet werden.

Auf den Datenseiten liegen die Einträge nicht nach dem Index geordnet, sondern in historisch gewachsener Anordnung im Anfangsbereich der Datenseite. Die Ordnung bezüglich des Primärschlüssels wird durch eine Positionsliste hergestellt, die sich am Ende einer jeden Datenseite befindet. Diese Positionsliste ist von rechts nach links geordnet, sodass Positionsliste und Daten aufeinander zu wachsen. Wird nun ein Datensatz gesucht, so kann er anhand der Positionsliste gefunden und ausgelesen werden. Abbildung 5.5 zeigt schematisch den Aufbau einer Datenseite.

Soll nun mittels einer Anfrage ein Datensatz der Tabelle gelesen werden, so wird diese Anfrage nur dann vom Index optimal unterstützt, wenn in der Where-Bedingung genau nach den Feldern gesucht wird, die von diesem Index indiziert sind. Da in der MaxDB ein B*-Baumindex für jeden Primärex erstellt wird, könnte eine Anfrage in diesem Beispiel wie folgt lauten:

```
Select * from Einwohner where Stadt = ,Hamburg'
```

Zugriff auf einen
Datensatz

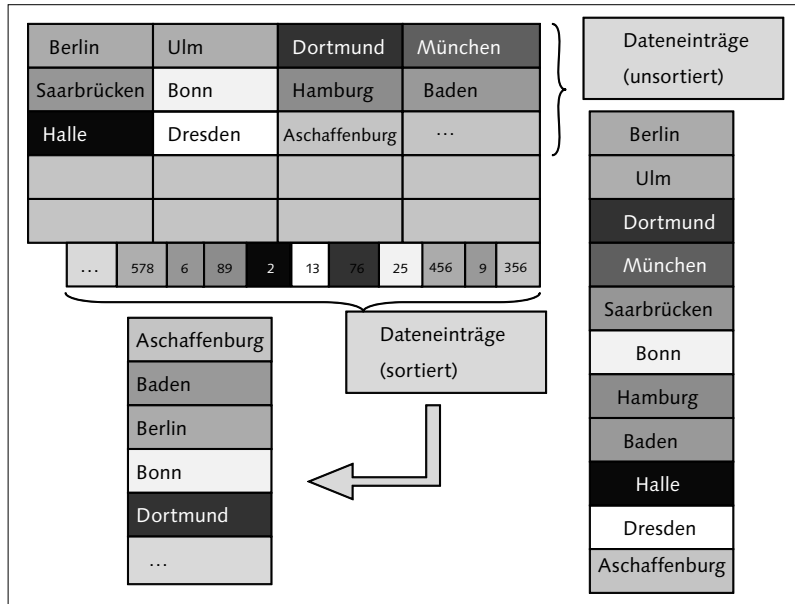


Abbildung 5.5 Aufbau einer Datenseite

Abbildung 5.6 soll den Zugriff auf einen Datensatz mit Hilfe eines Primärindizes veranschaulichen. Zunächst wird die Wurzelseite durchschritten. Sobald der gesuchte Wert kleiner als der Eintrag in der Wurzelseite ist, wird dem Verweis, der vor diesem Eintrag steht, auf die nächste Indexebene gefolgt. Der erreichte Knoten auf dieser Ebene wird nun nach demselben Prinzip durchsucht. Ist das Ende der Seite erreicht, ohne dass ein Eintrag gefunden wurde, der logisch größer als der Suchbegriff ist, so wird der letzte Verweis auf dieser Seite benutzt. Dies geschieht so lange, bis die Blattebene erreicht ist und der gesuchte Wert mittels der bereits besprochen Positionsliste auf den Datenseiten gefunden ist.

B*-Bäume für LONG-Werte

Für die Speicherung von Feldinhalten des Typs LONG werden, je nach ihrer Länge, eigene B*-Bäume verwaltet. Dabei werden zwei Typen von LONG-Werten unterschieden: kurze, die auf einer logischen Seite Platz finden, und lange LONG-Werte, deren Platzbedarf eine logische Seite übersteigt. Alle kurzen LONG-Werte werden in einem B*-Baum verwaltet, sodass auf der Datenseite der Tabelle statt dem Wert des LONG-Feldes, ein Verweis in diesen B*-Baum der kurzen LONG-Werte zeigt. Ist der Inhalt des LONG-Feldes größer als eine logische Seite, so wird ein eigener B*-Baum für diesen Wert an-

gelegt, und der Eintrag auf der Datenseite verweist auf den B*-Baum dieses einen Wertes. Abbildung 5.7 zeigt schematisch dieses Prinzip.

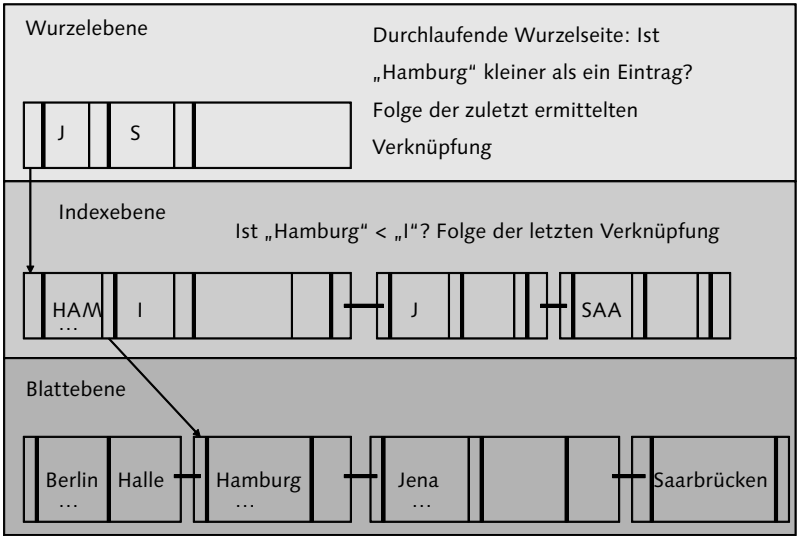


Abbildung 5.6 Zugriff auf einen Datensatz

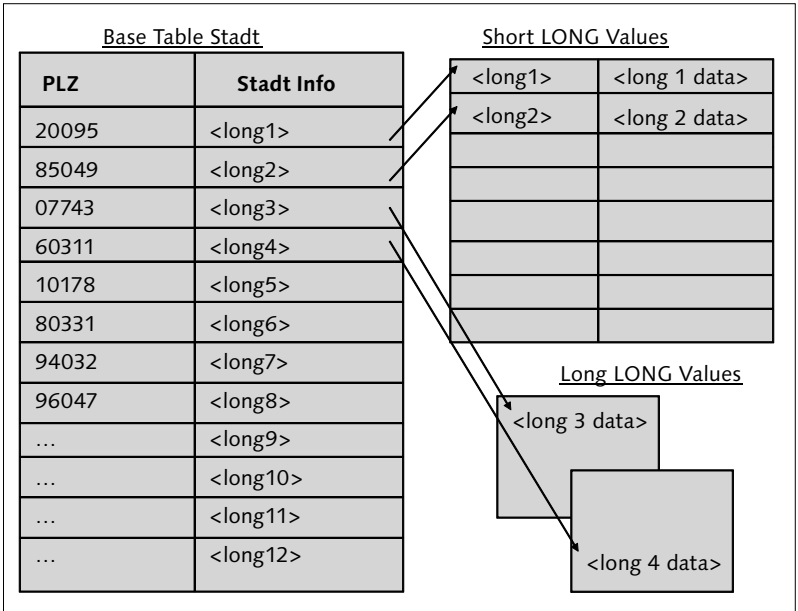


Abbildung 5.7 Speicherung von LONG-Werten

B*-Bäume für zusätzliche Indizes

Für jede Tabelle in der MaxDB werden die gerade besprochenen Indizes automatisch angelegt. Es werden also für den Primary Key einer Tabelle und für die LONG-Werte entsprechenden B*-Bäume erstellt. Natürlich können Sie auch über weitere Spalten einer Tabelle Indizes hinzufügen. Dies wird häufig für sogenannte *Secondary Keys* praktiziert, da bei dieser Modellierung Tabellen über diese Schlüssel mit anderen Tabellen logisch verknüpft werden. Diese logische Verknüpfung würde sich äußerst negativ auf die Performance auswirken, wenn nicht zusätzlich die Zugriffe auf Datensätze über Secondary Keys und damit B*-Bäume unterstützt würden. Prinzipiell ist der Aufbau eines B*-Baums für zusätzliche Indizes identisch mit dem Aufbau von B*-Bäumen für Primary Keys.

Invertierte Listen

Ein Unterschied ergibt sich jedoch aus der relationalen Modellierung von Tabellen. Jeder Datensatz wird durch das Feld bzw. die Felder des Primary Keys eindeutig identifiziert. Mit dieser Bedingung arbeiten auch die Indizes. Bei Secondary Keys ist die Bedingung nicht gegeben. Dies möchten wir Ihnen anhand des Beispiels von Adressdaten verdeutlichen.

Beispiel

Tabelle 5.1 hat die Postleitzahl als Primary Key und den Namen der Stadt, sowie eine Beschreibung als weitere Felder. Diese Tabelle wurde absichtlich so einfach gewählt und nur jede Stadt einmal aufgeführt, obwohl größere Städte natürlich mehrere Postleitzahlen besitzen.

PLZ	Ort	...
01067	Dresden	...
04103	Leipzig	...
06108	Halle	...
15230	Frankfurt	...
10115	Berlin	...
20095	Hamburg	...
28195	Bremen	...
30159	Hannover	...
33790	Halle	...
37620	Halle	...
40210	Düsseldorf	...

Tabelle 5.1 Beispiel für Datensätze mit gleichen Ortsnamen und unterschiedlichen Postleitzahlen

PLZ	Ort	...
44135	Dortmund	...
45127	Essen	...
50667	Köln	...
60311	Frankfurt	...
70173	Stuttgart	...
80331	München	...
90402	Nürnberg	...

Tabelle 5.1 Beispiel für Datensätze mit gleichen Ortsnamen und unterschiedlichen Postleitzahlen (Forts.)

Soll nun zusätzlich der Zugriff auf die Datensätze dieser Tabelle mit einem Index über dem Feld »Stadt« unterstützt werden, so können für einen Stadtnamen mehrere Postleitzahlen existieren, da es natürlich mehrere Städte mit den gleichen Namen gibt. Daher werden in diesem Fall invertierte Listen, wie in Tabelle 5.2 beispielhaft dargestellt, verwendet, die, solange sie auf eine Datenseite passen, flach abgespeichert werden.

Ort	PLZ
Berlin	10115
Bremen	28195
Dortmund	44135
Dresden	01067
Düsseldorf	40210
Essen	45127
Frankfurt	15230,60311
Halle	06108,33790,37620
Hamburg	20095
Hannover	30159
Leipzig	04103
München	80331
Nürnberg	90402
Stuttgart	70173

Tabelle 5.2 Invertierte Liste für den Index über der Spalte »Stadt«

Eindeutiges
Suchkriterium

Damit hat dieser B*-Baum für den zusätzlichen Index ein eindeutiges Suchkriterium. Wird nun diese invertierte Liste für einen Ort zu lang, so wird diese Liste ausgelagert und in einem eigenen B*-Baum verwaltet. Im ursprünglichen Index, der die invertierten Listen verwaltet, wird dann im Datenbereich für diesen Ort ein Verweis auf den für diese invertierte Liste neu angelegten B*-Baum angelegt.

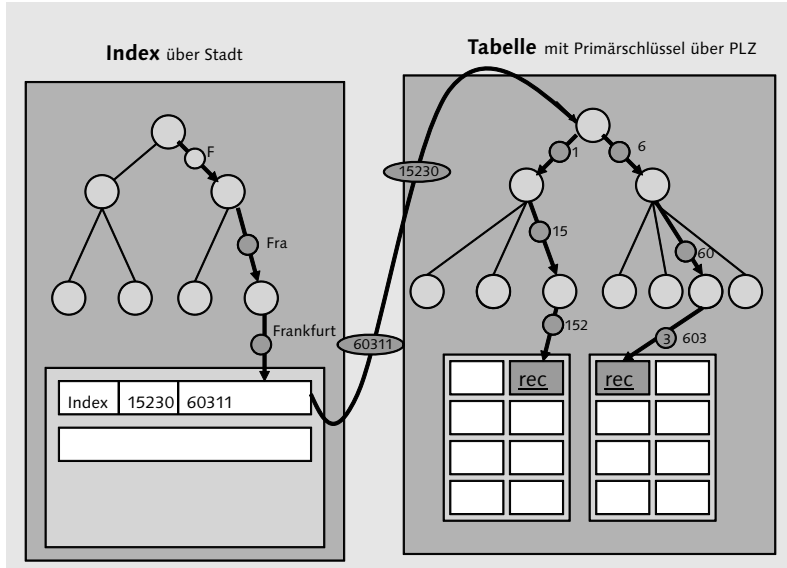


Abbildung 5.8 Zusätzlicher Index

Wichtig!

Daten werden in der MaxDB grundsätzlich nur im B*-Baum des Primary Keys gespeichert. In einem B*-Baum eines sekundären Index werden in den invertierten Listen die Werte nicht, wie im Beispiel die Nachnamen, noch einmal gespeichert, sondern immer Verweise auf den Primary Key angelegt. Diese Verweise enthalten dabei den *kompletten* Primary Key des Datensatzes, auf den gezeigt werden soll. Diese Tatsache spielt bei der Auswahl der Zugriffstrategie und damit bei der Beschleunigung von Datenzugriffen eine wichtige Rolle.

Ausführungskosten

Wie wichtig eine optimale Unterstützung von Anfragen durch performante, also selektive Indizes ist, zeigt der Vergleich der Ausführungskosten. Diese können ohne eine Indexunterstützung in einigen Fällen über 1000 Mal teurer sein. Umgekehrt gilt, dass ein teures SQL-Statement durch die Optimierung mit Indizes und/oder Ände-

rungen des Statements selbst möglicherweise ebenso auf ein Tausendstel reduziert werden kann. Jedoch möchten wir darauf hinweisen, dass zusätzliche Indizes natürlich Ressourcen benötigen, da sie bei Änderung an den Daten gepflegt und ebenso im Data Cache gehalten werden müssen. Deshalb sollten zuerst das Statement und der Code der Applikation daraufhin überprüft werden, ob sich nicht an dieser Stelle das Problem bereits lösen oder stark entschärfen lässt.

5.3 Der Datenbankoptimierer

Die Pflege und Bereitstellung von effektiven Indizes ist für performante Abfragen grundsätzlich wichtig. Ob ein Index verwendet wird oder – falls mehrere Indizes existieren – welcher Index für die Datensuche verwendet wird, entscheidet ein Programm in der Datenbank, der *Optimierer*. Von den Vorgängen bei einer Anfrage an die Datenbank kann die Performanz zu einem großen Teil abhängen. Um Ihnen diese Vorgänge zu verdeutlichen, stellen wir in diesem Abschnitt zuerst den Datenbankoptimierer vor, welcher oft auch *SQL Query Optimizer* genannt wird. Dazu werden wir die grundlegenden Eigenschaften des Optimierers erläutern und darstellen, nach welchen Kriterien Indizes beurteilt werden. Im Anschluss daran werden wir Ihnen anhand von charakteristischen Beispielen für SQL-Abfragen die wichtigsten Strategien vorstellen und erläutern, warum diese vom Optimizer ausgewählt werden.

5.3.1 Grundlagen

Der Ausführungsplan wird von einem Datenbankprogramm, dem Datenbankoptimierer, erstellt. Generell werden hier zwei Typen unterschieden: der *Rule Based Optimizer* (RBO) und der *Cost Based Optimizer* (CBO). Da von den Datenbanksystemen, die für den Einsatz mit SAP zertifiziert sind, lediglich Oracle die Möglichkeit des Einsatzes eines RBO bietet, während die übrigen ohne Ausnahme den CBO benutzen, zeigen wir Ihnen im Folgenden die Schritte und Verhaltensweisen des CBO.

Optimierertypen

Der CBO entscheidet, mit welcher Strategie auf die Daten zugegriffen wird. Im ersten Schritt werden dafür alle möglichen Zugriffsstrategien und im zweiten Schritt deren Kosten ermittelt, die aus der Anzahl von Seitenzugriffen bestehen. Für die Entscheidung über die Be-

Vorgehensweise

nutzung eines Indexes dienen unter anderem folgende Kriterien als Grundlage:

Kriterien für die
Verwendung von
Indizes

- ▶ **Speicherung auf dem physikalischen Medium**
Wie effektiv ein Index ist, hängt von der Verteilung der Daten auf dem Speichermedium ab. Sind diese stark verteilt, so sind mehr langsame Lesezugriffe nötig, als wenn mit einem Lesezugriff mehrere benötigte Daten gelesen werden könnten.
- ▶ **Verteilung der Feldinhalte**
Auch die Verteilung der gesuchten Feldinhalte innerhalb einer Tabelle wird vom Datenbankoptimierer berücksichtigt, da es für die Entscheidung wichtig ist, ob die Inhalte gleichmäßig über die Tabelle verteilt oder in Clustern abgespeichert sind.
- ▶ **Anzahl der unterschiedlichen Werte indizierter Felder**
Je mehr unterschiedliche Werte ein indiziertes Feld besitzt, desto wirksamer ist der betreffende Index und desto höher ist seine Selektivität. Mit *Selektivität* bezeichnet man also die Anzahl der unterschiedlichen Werte einer Spalte im Verhältnis zur Gesamtanzahl. In der Literatur ist zu finden, dass der Datenbankoptimierer nur Indizes benutzt, wenn dadurch die zu durchsuchende Datenmenge auf 5–10% reduziert wird.
- ▶ **Größe der Tabellen**
Wenn die Tabellen eine sehr geringe Größe aufweisen, dann kann es günstiger sein, die Tabelle komplett zu scannen, da somit weniger Lesezugriffe (also Kosten) entstehen.

Verwendung von Optimizer-Statistiken

Optimizer-Statistiken werden vom SQL Database Optimizer nur bei Joins oder Operationen auf Views für die Wahl der idealen Ausführungsstrategie verwendet. Views bestehen typischerweise aus der Verknüpfung von mehreren Tabellen anhand bestimmter Spalten, sie stellen also technisch gesehen auch Joins dar.

Optimizer-
Statistiken

Diese Informationen für die Optimizer-Statistiken werden zum Teil selbstständig von der Datenbank im internen Dateiverzeichnis vorgehalten. Die Erstellung und Aktualisierung weiterer statistischer Informationen über die vorhandenen Datenbanktabellen muss vom Datenbankadministrator initiiert werden, woraufhin sie dann im Datenbank-Katalog gespeichert werden. Diese Statistiken sollten Sie normalerweise mindestens einmal pro Woche ausführen, spätestens

aber dann, wenn sich die Inhalte einer Tabelle signifikant geändert haben. Diese statistischen Informationen können Sie mithilfe des Database Manager GUI sowie direkt auf der Kommandozeile manuell oder automatisch durchführen. Es ist dabei zu beachten, dass nur die ersten 1022 Byte eines Spaltenwertes betrachtet werden, sodass es die Möglichkeit kleiner Unschärfen gibt, wenn Spaltenwerte in den ersten 1022 Bytes übereinstimmen.

Im DBMGUI können Sie diese Statistiken für einzelne oder für alle notwendigen Tabellen erstellen – sowie für alle Tabellen, für die es überhaupt möglich ist. Abbildung 5.9 zeigt den Dialog, in dem Sie diese Einstellungen vornehmen können.

Manuell – DBM
GUI

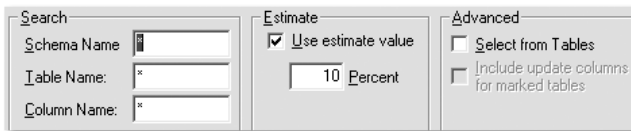


Abbildung 5.9 Einstellungen für die Aktualisierung der Optimizer-Statistiken im DBMGUI

Um zum oben abgebildeten Bildschirm zu gelangen und die Aktualisierung der Optimizer-Statistiken durchzuführen, sind folgende Schritte nötig:

1. Konnektieren Sie sich im DBMGUI zur Datenbankinstanz.
2. Wählen Sie INSTANCE • TUNING • OPTIMIZER STATISTICS aus.
3. Wählen Sie die gewünschten Tabellen aus.
4. Starten Sie die Suche durch Auswahl von SEARCH im Menüpunkt ACTIONS.
5. Konfigurieren Sie den Update-Vorgang.
6. Starten Sie die Aktualisierung durch ACTIONS • EXECUTE.

Die drei Spalten SEARCH, ESTIMATE und ADVANCED dienen zur Konfiguration des Update-Vorgangs der Optimizer-Statistiken. Wenn Sie den Vorschlag übernehmen, werden alle Tabellen aufgelistet, für die ein Update nötig wäre.

Auswahl der zu
aktualisierenden
Tabellen

Möchten Sie hingegen alle Tabellen anzeigen, für die ein Update möglich ist, so müssen Sie unter ADVANCED die Option SELECT FROM TABLES aktivieren. Soll dies nur für einzelne Tabellen ausgeführt wer-

den, so kann unter SEARCH nach dieser oder einer einzelnen Spalte gesucht werden.

Konfiguration des Update-Vorgangs

Je nach Größe der Tabellen und nach dem Verteilungsgrad müssen Sie eventuell den Umfang der Stichprobe in der Spalte ESTIMATE verändern. SAP empfiehlt, ab einer Größe von 1 Million Datensätze die Stichprobe auf 20% zu setzen, um ein genügend sicheres Ergebnis zu erhalten. In seltenen Fällen kann es auch nötig sein, die Stichprobengröße auf 100% zu vergrößern. Möchten Sie eine Tabelle aus dem Update-Lauf ausschließen, so ist dies durch einen Wert von 0% in diesem Feld möglich.

Einplanung im DBMGUI

Wie bereits erwähnt, haben Sie auch die Möglichkeit, die Aktualisierung der Optimizer-Statistiken automatisch einzuplanen. Abbildung 5.10 zeigt den Bildschirm, in dem Sie dies einstellen können.

Führen Sie dazu die folgenden Schritte aus:

1. Konnektieren Sie sich im DBMGUI zur Datenbankanstanz
2. Wählen Sie INSTANCE • AUTOMATIC STATISTICS UPDATE... aus.
3. Klicken Sie auf den ON-Button.

Damit werden für die Spalten und auch Tabellen, die in der Systemtabelle SYSUPDSTATWANTED eingetragen sind, *eventgesteuert*, also die Optimizer-Statistiken automatisch aktualisiert.

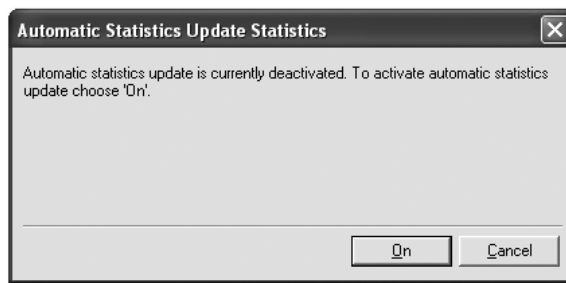


Abbildung 5.10 Automatische Aktualisierung der Optimizer-Statistiken im DBMGUI

Manuell – SQL-Statement

Natürlich können Sie diese Funktionen auch auf der Kommandozeile durchführen. Das `update_statistics_statement` hat dabei die folgenden Parameter:

Parameter	Beschreibung
Schema_name	Name des Datenbankschemas
Table_name	Tabellenname einer Basistabelle
Column_name	Spaltenname
Sample_Definition	ESTIMATE <Sample_Definition> ::= SAMPLE <unsigned_integer> ROWS SAMPLE <unsigned_integer> PERCENT
As per System Table	Bewirkt, dass die Statistiken für alle Tabellen, die in der Systemtabelle SYSUPDSTATWANTED verzeichnet sind, aktualisiert werden.
Identifier	Name einer Basistabelle

Tabelle 5.3 Parameter des update_statistics_statement

Beachten Sie bei diesem Statement, dass ein Benutzer nur Tabellen und Felder aktualisieren kann, für die er auch Zugriffsrechte besitzt. Ist dies geschehen, so können wir diese Statistikwerte aus der Systemtabelle OPTIMIZERINFORMATION selektieren. Dabei steht jede Zeile für die Statistikwerte von Indizes, Spalten oder Größe einer Tabelle.

Um die Optimizer Statistics für alle Basistabellen zu aktualisieren, führen Sie die folgenden Schritte aus:

1. Konnektieren Sie sich zur Datenbank mit:

```
/opt/sdb/programs/bin/dbmcli -u <SYSDBA-  
Benutzer>,<Passwort> -d <Datenbank> [-n <Datenbankhost>]
```

2. Aktualisieren Sie die Statistiken aller Tabellen:

```
UPDATE STATISTICS *
```

Sie können die Menge der zu analysierenden Datensätzen pro Tabelle von Hand steuern, indem Sie die SAMPLE_DEFINITION mit dem Parameter Estimate einstellen. Dabei können Sie konfigurieren, wie viele Tabellenzeilen bzw. wie viel Prozent der Tabellen bzw. Spaltenwerte gescannt werden. Wird keine SAMPLE_DEFINITION angegeben, so werden Zufallswerte benutzt.

Feintuning des
Aktualisie-
rungslaufes

Die Größe des Samples kann die Laufzeit des Aktualisierungslaufes stark beeinflussen. Wird dieser Parameter nicht mit angegeben, so wird die Größe dieser Sample-Probe aus der Definition der Tabelle ausgelesen. Das bedeutet, dass Sie bei der Erstellung von Tabellen auch diesen Aspekt berücksichtigen sollten, da er für die Leistungsfä-

higkeit der Datenbank mit entscheidend ist. Aufgrund der Tatsache, dass sich die Tabellen und deren Nutzung im Laufe der Zeit ändern können, können Sie diesen Wert auch im Nachhinein mit der Anweisung `Alter Table` verändern bzw. korrigieren. Eine Tabelle kann komplett vom Optimierungslauf ausgeschlossen werden, indem die Stichprobengröße mittels der `Alter-Table`-Anweisung mit 0 belegt wird. Wenn der `Estimate`-Parameter überhaupt nicht angegeben wird, so hat dies einen kompletten Scan der Tabelle zur Folge, was bei umfangreichen Tabellen natürlich auch zu langen Laufzeiten führt.

Wird die Option `Update-Statistics-AS-PER-SYSTEM-TABLE` verwendet, so werden, wie in der Variante mit dem DBMGUI, die Statistiken derjenigen Tabellen aktualisiert, die in der Systemtabelle `SYSUPDSTATWANTED` vermerkt sind. Nach einem erfolgreichen Lauf mit dieser Option werden die Tabellennamen aus dieser Systemtabelle gelöscht.

Automatisch – SQL-Statement

Um die Aktualisierung der Optimizer-Statistiken automatisch mithilfe der Kommandozeile einzuplanen, kann das Statement `auto_update_statistics` verwendet werden.

1. Konnektieren Sie sich zur Datenbank mit:

```
/opt/sdb/programs/bin/dbmcli -u <SYSDBA- Benutzer>,  
<Passwort> -d <Datenbank> [-n <Datenbankhost>]
```

2. Starten Sie die automatische, eventgesteuerte Aktualisierung:

```
auto_update_statistics <mode>
```

Für die Aktualisierung haben Sie drei Modi zur Auswahl:

- ▶ **On:** Schaltet die automatische Aktualisierungsfunktion ein. Bitte beachten Sie, dass dies wieder eventgesteuert auf Basis der schon öfter erwähnten Systemtabelle `SYSUPDSTATWANTED` geschieht. Da auch dieses DBM-Kommando einen eigenen Event Task benötigt, stellen Sie sicher, dass der Datenbankparameter `_MAXEVENTTASKS` groß genug gewählt wird.
- ▶ **Off:** Schaltet die automatische Aktualisierungsfunktion aus.
- ▶ **Show:** Liefert Ihnen den aktuellen Status der automatischen Aktualisierungsfunktion zurück; mögliche Werte sind hierbei:
 - ▶ **On:** Die automatische Aktualisierungsfunktion ist aktiv.
 - ▶ **Off:** Die automatische Aktualisierungsfunktion ist abgeschaltet.
 - ▶ **Unknown:** Der Status der automatischen Aktualisierungsfunktion konnte nicht festgestellt werden.

5.3.2 Kriterien für die Auswahl bestimmter Zugriffsstrategien

Aktuelle Optimierer-Statistiken sind für eine korrekte Auswahl der Zugriffsstrategie durch den Optimierer nur bei Join-Operationen wichtig. Im folgenden Abschnitt möchten wir Ihnen einige signifikante Beispiele für Abfragen darstellen und die Gründe aufzeigen, warum die jeweilige Zugriffsstrategie gewählt wird.

Die Wahl der Zugriffsstrategie hängt von vielerlei Faktoren ab:

Faktoren

- ▶ Wie ist die Abfrage geartet, also nach welchen Spalten wird in der Where-Klausel differenziert?
- ▶ Existieren Indizes, und welche Selektivität weisen diese auf?

Dies alles berücksichtigt der Optimizer bei der Auswahl der Zugriffsstrategie.

Die Beispieltabelle

Als Beispiel soll eine Tabelle (Table) mit 7 Spalten (Column1 bis Column7) dienen, die einen Primary Key über 3 Spalten (Column1, Column2, Column3), und einen zusätzlichen Index über der 5. Spalte (Column5) besitzt. Die Spalten des Primary Keys sind von unterschiedlicher Selektivität. Column1 weist eine sehr geringe Selektivität auf, Column3 dagegen eine sehr hohe. Column2 liegt in der Mitte mit einer mittleren Selektivität. Column5, über welcher ein zusätzlicher Index angelegt ist, hat ebenso wie Column3 eine sehr hohe Selektivität.

Zugriff über Primary Key

Generell ist es immer ratsam, bei Abfragen auf Tabellen alle Felder des Primary Keys in der Abfrage zu benutzen:

```
select * from table where Column1 = ‚Jürgen‘ AND Column2 = ‚Meier‘ AND Column3 = ‚12.10.1970‘
```

Diese Abfrage wird mit der Ausführungsstrategie *Equal Condition for Key Column* ausgeführt, also mit Zugriff auf den oder die gewünschten Datensätze über den Primary Key. Da die Daten auch physikalisch in Reihenfolge des Primary Keys abgelegt sind, bietet der Primary Key eine sehr gute Unterstützung für Abfragen, die nicht alle Felder des Primary Keys benutzen.

Equal Condition
und Range
Condition

```
select * from table where Column1 = ‚Jürgen‘ AND Column2 = ‚Meier‘
```

Die Ausführung dieser Abfrage geschieht ebenfalls über den Primary Key. Durch die physikalische Anordnung der Daten gemäß dem Primary Key kann in diesem Fall der Zugriff über die ersten beiden Schlüsselfelder erfolgen, und die gewünschten Datensätze können bereits auf dem Primary Key Index identifiziert werden, der ja alle Felder des Primary Keys enthält. Die Strategie, die dieses Verhalten implementiert, heißt *Range Condition for Key Column*.

Primary Key versus Index

Nur ein Mandant Der gerade angesprochene Ausführungsplan ist jedoch nicht zwangsläufig effektiv. Im SAP-Umfeld ist der Mandant in sehr vielen Tabellen ein Teil des Primary Keys. Wenn nun ein System, wie dies in BI sehr häufig der Fall ist, nur einen Mandanten besitzt, so kann die folgende Abfrage nach allen Benutzern aus dem Mandanten »800« mit Straße »Holzweg« zum Durchsuchen der gesamten Tabelle führen:

```
select * from table where Column1 = ‚800‘ AND Column4 = ‚Holzweg‘
```

Bei dieser Abfrage wird die *Range Condition for Key Column* als Strategie verwendet, jedoch müssen alle Datensätze der Tabelle durchsucht werden. Diese Abfrage könnte deutlich beschleunigt werden mit einem zusätzlichen Index über der Spalte `Column4`, der vermutlich eine hohe Selektivität aufweisen würde. Ein großer Vorteil eines Indexes über `Column4` ist die Struktur von Sekundärindizes. Dabei können die im Sekundärindex gespeicherten Werte des Primary Keys für die Selektion der Daten verwendet werden. Würde ein Sekundärindex über der `Column4` angelegt, so würde dies dazu führen, dass bei diesem Beispiel die Zugriffstrategie sich nicht des Primary Keys bedienen würde, sondern der Zugriff über den Index über `Column4` mit der Strategie *Equal Condition for Indexed Column* erfolgt.

Es besteht auch die Möglichkeit, dass der Index über `Column4` trotz seiner angenommenen schlechten Selektivität und der sehr hohen Selektivität der Spalte `Column1` trotzdem für den Zugriff verwendet wird. Dies geschieht dann, wenn bereits bei der Überprüfung der verschiedenen Zugriffsstrategien herausgefunden wird, dass der ge-

suchte Wert in `Column4` nicht existiert und die Ergebnismenge dementsprechend leer ist.

Zugriffsstrategien

Wir haben in diesem Kapitel bisher zwei Strategien unterschieden: Die Strategie *Equal Condition for Index Column* ist eine Suchstrategie, die eine Auswertung anhand einer Vergleichsoperation vornimmt, dabei aber eine invertierte Liste nutzt. Die Einträge in den Tabellen werden direkt angesprochen. Bei der Strategie *Range Condition for Key Column* wird eine sequenzielle Suche auf Teilen der Tabelle durchgeführt. Neben den hier besprochenen Suchstrategien existieren noch weitere Strategien, die sich durch das `Explain`-Statement anzeigen lassen können.

Index versus Full Table Scan

Ein *Full Table Scan* wird grundsätzlich verwendet, wenn keine ausreichende Unterstützung der Abfrage durch den Primary Key oder weitere Indizes zur Verfügung steht. Des Weiteren wird er auch benutzt, falls die Tabelle sehr klein ist und daher weniger Seiten geladen werden müssen als bei einem Zugriff über einen Index. Schließlich verursacht der Zugriff auf einen Index auch Kosten, und bei kleinen Tabellen müssen dann zusätzlich alle Datensätze durchsucht werden.

Bei der Unterstützung von Abfragen kann ein Full Table Scan oftmals umgangen werden, wenn für die Abfrage nur Felder benutzt werden, die für sich betrachtet eine sehr geringe Selektivität aufweisen:

Full Table Scan
umgehen

```
select * from table where Column4 = ‚Finanzbuchhaltung‘ AND
Column6 = ‚Teamleiter‘
```

Eine Möglichkeit, wie Sie die Ausführung dieses Statements stark beschleunigen könnten, wäre durch einen zusammengesetzten Index über den Spalten `Column4` und `Column6`. Da zwar jede Spalte für sich genommen eine geringe Selektivität aufweist, diese in Kombination jedoch ein akzeptables Entscheidungskriterium darstellen, kann dieser Index eine genügend hohe Selektivität bereitstellen, um bei der Verwendung zum Zugriff auf die Daten einen Performance-Gewinn gegenüber dem Full Table Scan zu erzielen. Bei Tabellen geringerer Größe können Sie dies wie folgt herausfinden:

1. Öffnen Sie einen SQL-Dialog über das SQL Studio oder per `dbmcli`.
2. Geben Sie nun folgendes Statement ein:

```
Select distinct Column4,Column6 from table
```

Das Statement liefert Ihnen alle Kombinationen der Werte der beiden Spalten Column4 und Column6. Wenn diese Ergebnismenge sehr viele Werte enthält, so ist anzunehmen, dass ein Index über diesen Spalten eine ausreichende Selektivität erzielt.

Joins

Optimizer-
Statistiken
entscheidend

Mit *Joins* werden Datenbankabfragen bezeichnet, die mehrere Tabellen anhand der Werte einer oder mehrerer Spalten verbinden. Eine Darstellung der Ausführungsstrategien für Joins oder Abfragen auf Datenbank-Views (äquivalent zu einer Join-Abfrage) würde den Rahmen dieses Buches sprengen. Wir weisen nochmals darauf hin, dass bei der Auswahl der Ausführungsstrategie die Optimizer-Statistiken eine entscheidende Rolle spielen. Während die Statistiken für den Zugriff auf Basistabellen nicht verwendet werden, bilden sie eine wichtige Grundlage für die Entscheidung über die Ausführungsstrategie von Joins. Sollten Sie bei der Analyse von Joins oder Abfragen auf Datenbank-Views auf unerwartete Ausführungsstrategien treffen, so können veraltete Optimizer-Statistiken der Grund sein. In diesem Fall aktualisieren Sie diese für alle durch den Join oder die View benutzten Tabellen.

Indexes über Join-
Spalten wichtig

Des Weiteren ist es grundsätzlich ratsam, diejenigen Spalten der Tabellen, die Sie für einen Join benutzen möchten, mit einem Index auszustatten, der eine ausreichende Selektivität besitzt. Falls Ihnen dies nicht möglich ist, können Sie einen Index auch über mehreren Spalten erstellen, der dann durch die Kombination eine genügend hohe Selektivität erhält. Danach müssten Sie allerdings die Join-Bedingung an den neuen Index anpassen. Eine absolute Lösung für diese Problematik lässt sich leider nicht angeben, da es für jedes Problem meistens mehrere, oftmals ganz individuelle Lösungsansätze gibt.

5.4 Caches

Für die hohen Zugriffsgeschwindigkeiten heutiger Datenbanksysteme sind u. a. die verwendeten Caching-Strategien auf Datenbankebene verantwortlich. Eine fehlerhafte Konfiguration dieser Caches kann auf die Performance sehr negative Auswirkungen haben. Deshalb stellen wir in diesem Abschnitt nochmals die verschiedenen Ca-

ches der MaxDB und deren Verwendung sowie die Analyse der optimalen Trefferrate dar. Schließlich besprechen wir noch die Problematik der richtigen Cache-Größe.

5.4.1 Motivation

»Disk access is excruciatingly slow«. Diese Aussage aus *Database – Principles, Programming and Performance* von O'Neil aus dem Jahre 2001 trifft genau das Kernproblem, das für die Existenz von Caches verantwortlich ist. Damit Daten von einer Festplatte gelesen werden können, müssen zuerst die Schreib-/Leseköpfe auf die richtige Spur gesetzt werden. Dies wird als *Seek Time* bezeichnet. Im Anschluss daran muss der Lesekopf warten, bis er sich aufgrund der Drehung über der richtigen Seite befindet. Diese Zeitspanne wird im Allgemeinen *Latenzzeit* genannt. Darauf folgt die wirkliche Lesezeit, oft auch *Transferzeit* genannt, in der die gewünschten Blöcke gelesen werden. Da es sich bei all diesen Vorgängen um mechanische Aktionen handelt, ist die Zugriffszeit im Vergleich zu der eines Hauptspeichers so »quälend« langsam.

Warum Caches?

Im Größenvergleich ergibt sich dabei für das Lesen einiger Tausend Bytes die Zeit von ca. 0,003 Sekunden. Im Vergleich dazu kann die gleiche Menge an Daten aus dem Hauptspeicher in ca. 0,00000001 Sekunden geladen werden. Aus diesem Grund ist es natürlich vorteilhaft, Daten, die öfter gebraucht werden, im Hauptspeicher, in Caches, zu belassen. Der Hauptspeicher gewährleistet jedoch keine persistente Speicherung der Daten, da diese bei einem Stromausfall oder beim einfachen Abschalten des Rechners verloren gehen. Da der verfügbare Platz auf dem Hauptspeicher sehr viel geringer ist als auf dem Festplattenspeicher, stellt sich das Problem der optimalen Verteilung des Hauptspeichers auf die verschiedenen Anwendungen.

0,003 sec versus
0,00000001 sec

5.4.2 Die verschiedenen Caches

Generell benutzt MaxDB drei Caches: *I/O Buffer Cache*, *Catalog Cache* und *Log I/O Queue*. Alle Caches sind in Regionen unterteilt, um parallelen Zugriff zu ermöglichen und so die Schreibrate zu erhöhen. Beim Zugriff auf eine Region wird diese für die Benutzung durch einen anderen User Task gesperrt. Gibt es Kollisionen beim Zugriff auf die Regionen, führt dies zu Wartezeiten auf die Freigabe dieser Bereiche. Dies ist ein Anzeichen für eine hohe CPU-Belastung. Nor-

Drei Caches

malerweise werden diese Locks innerhalb von 1 Mikrosekunde wieder frei gegeben. Steht jedoch der Prozessor unter starker Last, so kann es sein, dass der Dispatcher des Betriebssystems diesem User Kernel Thread die CPU entzieht und der UKT weiterhin den Lock behält. Dadurch steigt natürlich das Risiko von Kollisionen bzw. Warteschlangen.

Data Cache

98% der Zugriffe
aus dem Cache

Durch große Data Caches werden in heutigen produktiven MaxDB-Installationen über 98% der Lese- und Schreibzugriffe aus dem Cache bedient. Da es wahrscheinlich ist, dass die Daten im Cache auch wieder geändert werden, ist es vorteilhaft, alle Änderungen an den Daten im Cache zu vollziehen und dies durch einen Eintrag im Redo Log persistent zu machen. In regelmäßigen Intervallen werden die Datensätze dann aus dem Data Cache in die Data Volumens und damit auf die Platte geschrieben. Wenn Daten im Cache nicht gefunden werden, wird die komplette Seite aus den Data Volumens gelesen und in den Data Cache geschrieben, um von dort weiter verwendet zu werden. Da der Zugriff auf Daten in den Data Volumens sehr langsam und deswegen teuer ist, ist eine maximale Data Cache Hitrate immer von Vorteil.

Trefferquote im
Data Cache

Eine Hit Rate von 99% oder mehr ist allerdings kein hinreichendes Kriterium, da durch die große Menge von Statements, die aus dem Data Cache bedient werden, eine unperformante Transaktion überdeckt wird. Wenn ein einziges Statement 10 Seiten mit 1000 Datensätzen laden muss, um einen Record auszulesen und dann die nächsten 990 Abfragen aus dem Cache erledigt werden können, so beträgt die Hitrate 99% – und dennoch ist dieses eine Statement nicht performant. Solange genügend physikalischer Hauptspeicher zu Verfügung steht, ist es ratsam, den I/O Buffer Cache so groß wie möglich zu gestalten, da sich die Lesezeiten von Daten in einem großen Cache nicht von denen in einem kleinen Cache unterscheiden, jedoch das »Risiko« von physikalischen Datenzugriffen vermindert wird.

Liegt die Data Cache Hitrate über einen längeren Zeitraum unter 99%, so kann dies mehrere Ursachen haben. In den meisten Fällen ist entweder der Cache zu klein und/oder es ist ein Hinweis auf ineffiziente SQL-Statements. Wie Sie die Ursache herausfinden, erfahren Sie in Abschnitt 5.5, »Analyse-Tools«.

Converter Cache

Da die Datenbank nur auf logischen Seiten arbeitet, ist ein Automatismus erforderlich, der die Zuordnung von logischen Pages zu physikalischen auf der Festplatte erledigt. Dies ist die Aufgabe des *Converters*. Die Zuordnungstabelle wird beim Starten der Instanz komplett in diesen Cache geladen. Die Größe dieses Caches kann also nicht konfiguriert werden, da er die nötige Größe beim Start automatisch zugewiesen bekommt. Sollte der Speicherbedarf während des Betriebes wachsen, weil dynamisch neue Data Volumes hinzugenommen wurden, so wird diesem Cache Speicher vom I/O Buffer Cache zugewiesen.

Converter Cache für Zuordnungstabelle

Catalog Cache

Im Catalog Cache werden Informationen der SQL-Statements gespeichert. Dies sind unter anderem Informationen über den Parse-Vorgang, Input-Parameter und Ausgabewerte. Diese Werte werden, falls der Parameter `SHAREDSQL` den Wert `yes` besitzt, spezifisch für jeden User abgespeichert. Wird ein gleiches SQL-Statement von unterschiedlichen Benutzern abgesetzt, so wird es, falls dieser Parameter aktiviert ist, auch mehrfach abgespeichert. Für jeden User Task wird ein spezifischer Bereich im Catalog Cache reserviert und wieder freigegeben, sobald die Benutzer-Session beendet ist. Sollte dieser Cache seinen maximalen Füllgrad erreicht haben, so werden die Informationen in den Data Cache verschoben. Der Catalog Cache sollte eine Trefferrate von über 90% aufweisen.

Catalog Cache für SQL-Statements

OMS Cache

Dieser Cache wird nur im Instanztyp MaxDB liveCache verwendet. In diesem Cache werden Daten in einer Heap-Datenstruktur, die aus mehreren, miteinander verbundenen Bäumen besteht, gespeichert und verwaltet. Dabei werden lokale Kopien der OMS-Daten vorgehalten, die in den Heap geschrieben werden, sobald zum ersten Mal auf einen konsistenten View zugegriffen wird. Das Datenbanksystem kopiert die Daten jeder OMS-Version in diesen Heap, sobald sie gelesen werden. Um nun ein persistentes Objekt zu lesen, sucht die MaxDB zuerst in diesem Heap. Wird es nicht gefunden, so wird der Data Cache durchsucht. Als letzter Ausweg werden die gesuchten Daten dann von der Data Area in den Data Cache und danach in den

OMS Cache für liveCache-Instanzen

HEAP geschrieben. Der HEAP dient dabei als Arbeitsplatz, an welchem die Daten verändert werden, um dann, sobald ein COMMIT gesetzt wird, zurück in den Data Cache geschrieben zu werden. Da dieser Buffer für liveCache-Instanzen sehr wichtig ist, sollte auch dieser im Rahmen der Hardwaremöglichkeiten großzügig mit Arbeitsspeicher versehen werden.

Log I/O Queue

Erhöhung der Performance

Um nicht bei jedem COMMIT die Daten verstreut über die Daten-Volumen schreiben zu müssen, was sich auf die Performanz der Schreibvorgänge negativ auswirkt, werden die Änderungen an den Daten in einem Redo Log gespeichert. Dieses Redo Log wird sequenziell beschrieben, was zu performanteren Schreibvorgängen führt. Da alle Änderungen an den Daten im Redo Log protokolliert werden, ist es wichtig, für dieses Log Volume die performantesten Platten zu benutzen. Um die Schreibvorgänge ins Redo Log zu beschleunigen, werden sie in Log Queues gecached. Durch den Parameter `MAX_LOG_QUEUE_COUNT` wird die maximale Anzahl der Log Queues definiert. Wie viele Queues wirklich verwendet werden, bestimmt entweder die Datenbank oder der Administrator mithilfe des Parameters `LOG_QUEUE_COUNT`. Die Größe in Pages zu 8 Kilobyte der Log Queue(s) wird durch den Parameter `LOG_IO_QUEUE` eingestellt.

Richtige Größe des Caches

Auch für diesen Cache besteht die generelle Problematik der richtigen Speichergröße. Er sollte groß genug sein, um die Spitzen der Schreibvorgänge im Redo Log puffern zu können. Mit dem im folgenden Abschnitt vorgestellten Database Analyzer kann ermittelt werden, ob es *Log Queue Overflows* gab. Dies bedeutet, dass die Log Queue voll ist, bevor die Daten auf die Log Volumes geschrieben werden konnten. Sollte das der Fall sein, so kommt es zu Performance-Engpässen. In diesem Fall überprüfen Sie die Geschwindigkeit der Hardware. Ist die Geschwindigkeit der Hardware für die zu verarbeitende Menge an Daten generell zu klein, so zögert eine Erweiterung der Log Queue die Overflow-Situation nur hinaus. Um diese Situation grundsätzlich zu verhindern, können Sie mit dem Parameter `MaxLogWriterTasks` die Anzahl der Task, welche die Daten parallel auf die Logvolumes schreiben, erhöhen. In Kombination mit unterschiedlichen Festplatten, auf denen die Log Volumes liegen, erhöhen Sie dadurch die Performanz und wirken Log Queue Overflows entgegen.

Eine Vergrößerung der Log Queue löst das Performanceproblem von Log Queue Overflows nur wenn die Hardware, welche die Log Volumens beherbergt, grundsätzlich schnell genug ist, und die Overflows durch einzelne Peaks der zur verarbeitenden Datenmenge entstehen.

Des Weiteren kann nachgelesen werden, wie viele Seiten der Log Queue bis zu diesem Zeitpunkt maximal benutzt wurden. Dies gibt Aufschluss über die Qualität der konfigurierten Größe der Log Queue. Liegt dieser Wert über einen langen Zeitraum deutlich unter der Anzahl der zur Verfügung stehenden Seiten im Cache, so kann durch eine Verkleinerung dieses Caches Hauptspeicher für andere Anwendungen oder Caches freigemacht werden. Es wird empfohlen, hierbei einen Sicherheitsspielraum für eventuelle Lastspitzen zurückzubehalten.

5.4.3 Die richtige Größe der Caches

Zu wenig Cache ist schlecht für die Performance der MaxDB. Als Faustregel kann der Wert von 66% des gesamten Hauptspeichers für die Caches dienen. Wird jedoch zu viel Cache konfiguriert, der auf der Hardware physikalisch nicht zur Verfügung steht, so führt dies zu Swaps, was unter allen Umständen vermieden werden sollte, da es die Performance des Systems verschlechtert. Der konfigurierte Cache, also der Speicherplatz im Hauptspeicher des Servers, wird von der MaxDB beim Starten, genauer zu Beginn der Phase »Admin«, allokiert und steht somit den übrigen Anwendungen nicht mehr zur Verfügung. Wird also zuviel Cache konfiguriert, so kann dies zu Speicherengpässen bei anderen Anwendungen führen. Generell gilt: Solange genügend Hauptspeicher vom System bereitgestellt wird, schadet ein zu großer Cache in keinem Fall. Die Dauer einer Suche nach einem Datensatz ist im Hauptspeicher völlig unabhängig von der Größe des Caches.

Grundlegende
Problematik

5.4.4 Die wichtigsten Informationen über Caches

In diesem kurzen Abschnitt möchten wir Ihnen eine Referenz an die Hand geben, die Sie schnell zu den gewünschten Informationen führt. Wir zeigen, wie Sie sowohl im SAP-System, im DBMGUI, als auch mittels des dbmcli an die wichtigsten Werte der Caches, wie deren Größen und Trefferraten, gelangen.

DB50 im SAP-System

Im SAP-System stellt Ihnen die Transaktion DB50 ein probates Mittel zur Verfügung, um sich einen schnellen und präzisen Überblick zu dem aktuellen Status der Caches zu verschaffen. Dies ist auch per DBMGUI möglich. Die Abfrage des Cache-Status per dbmcli ist leider nicht so bequem, soll aber als Möglichkeit dennoch gezeigt werden.

Caches in der DB50 ansehen

Die Transaktion DB50 (siehe Abbildung 5.11) stellt ausführliche Informationen zu den Caches und deren Auslastung bereit. Um zu den Daten zu gelangen, gehen Sie wie folgt vor:

1. Loggen Sie sich zunächst auf dem SAP-System ein.
2. Rufen Sie die Transaktion DB50 auf, um sich den aktuellen Status der MaxDB anzeigen zu lassen. Sie gelangen zunächst auf den Übersichtsbildschirm.
3. Öffnen Sie nun den Pfad **AKTUELLER STATUS • SPEICHERBEREICHE • CACHES**.

Datenbankserver:	G77DB1	Datum/Uhrzeit:	09.07.2008 17:23:06	
Datenbankname:	G77	Datenbankstart:	07.01.2008 15:51:38	

Cache-Größen			
	Größe in KB	Größe in Seiten	
I/O Buffer Cache	3.145.728	393.216	
Data-Cache	2.949.920	368.740	
Converter	184.848	23.106	
Sonstiges	10.960	1.370	
Catalog-Cache	80.000	10.000	
Sequence-Cache	8	1	

Cache-Zugriffe				
	Zugriffe	Erfolgreich	Nicht erfolgreich	Trefferrate
Data-Cache	6.120.018.117	6.087.869.330	32.148.787	99,47%
Undo	81.424.210	81.424.210	0	100,00%
OMS Data	0	0	0	100,00%
SQL Data	6.038.593.907	6.006.445.120	32.148.787	99,47%
Catalog-Cache	1.725.344.785	1.715.004.004	10.341.990	99,40%
Sequence-Cache	0	0	0	100,00%

Cache-spezifische Parametereinstellungen	
CACHE_SIZE	393216
CAT_CACHE_SUPPLY	10000

Abbildung 5.11 Übersicht über die Informationen der Caches

In der Übersicht sehen Sie nun im oberen Teil die Größe der Caches in Bytes und in Pages. Diese Werte sind sehr nützlich, da die Größe einiger Caches nicht explizit konfiguriert werden kann.

Caches im DBMGUI ansehen

Im DBMGUI können Sie die gleiche Menge von Werten (siehe Abbildung 5.12) finden wie gerade im Abschnitt zuvor in der Transaktion DB50. Der einzige Unterschied besteht in der Angabe der Cache-Größen. Hier im DBMGUI wird die Einheit Megabyte, auf zwei Nachkommastellen gerundet, verwendet. In der Transaktion DB50 sehen Sie die Werte in Kilobyte. Die auf den ersten Blick scheinbar verschiedenen Werte sind natürlich gleich; der Unterschied kommt durch Rundung und durch die Umrechnung zustande.

Einheit: Megabyte

Cache Sizes		Cache Accesses				
	Size		Accesses	Successful	Unsuccessful	Hit Rate
IO Buffer Cache	3.072,00 MB	Data Cache	6.120.041.535	6.087.892.711	32.148.824	99,47%
Data Cache	2.880,78 MB	SQL Data	6.038.616.012	6.006.467.188	32.148.824	99,47%
Converter	180,52 MB	OMS Data	0	0	0	100,00%
Misc.	10,70 MB	History/Undo	81.425.523	81.425.523	0	100,00%
Catalog Cache	78,13 MB	Catalog Cache	1.725.348.307	1.715.007.181	10.342.335	99,40%
Sequence Cache	0,01 MB	Sequence Cache	0	0	0	100,00%

Cache-Specific Parameter Settings	
	Value
Ⓞ CACHE_SIZE	3.072,00 MB
Ⓢ CAT_CACHE_SUPPLY	78,13 MB

Abbildung 5.12 Die wichtigsten Werte der Caches im DBMGUI

Um diese Informationen zu erhalten, sind folgende Schritte im DBMGUI nötig:

Schritte im DBMGUI

1. Konnectieren Sie sich per Doppelklick auf eine Instanz zur Datenbank.
2. Öffnen Sie nun die Übersicht der Caches über den Menüpfad INFORMATION • CACHES.
3. Mithilfe des Buttons REFRESH in der obersten Zeile können die Werte aktualisiert werden, da sich diese natürlich im Betrieb ändern können. Im Prinzip erhalten Sie im DBMGUI dieselben Daten wie in der Transaktion DB50.

Caches per dbmcli ansehen

Die Cache-Daten können Sie natürlich auch über die Kommandozeile mithilfe von dbmcli einsehen. Dies ist allerdings etwas aufwendiger, da die in den beiden vorigen Abschnitten gezeigten Daten vom Da-

Aufwändiger

tenbanksystem in Tabellen geschrieben werden und deshalb mithilfe von SQL-Befehlen abgefragt werden müssen. Dies gestaltet sich im dbmcli etwas unfreundlicher als im SQL Studio. Trotzdem möchten wir Ihnen diese Abfragen und deren Ergebnisse mithilfe des Tools dbmcli vorstellen. Das folgende SQL-Statement zeigt, dass sich ein Teil der Werte in der Tabelle IOBUFFERCACHES befindet. Da sich die Größen des Data Caches und des Converter Caches nicht explizit konfigurieren lassen, können diese Werte also nicht über die Ausgabe von Parametern erlangt werden, sondern müssen von der Datenbank mittels Tabellen bereitgestellt werden.

Um sich die Daten der Caches anzeigen zu lassen, gehen Sie wie folgt vor:

Caches im dbmcli ansehen

1. Verbinden Sie sich zur Datenbank:

```
/opt/sdb/programs/bin/dbmcli -d MAXDB -n <host> -
u <user>,<passwort>
```

2. Führen Sie den folgenden SQL-Befehl aus, der Ihnen die Cache-Daten ausgibt. Sie müssen das Statement nicht in Anführungszeichen setzen; es wird einfach hinter den Befehl `sql_execute` geschrieben.

```
/opt/sdb/programs/bin/dbmcli ON MAXDB> sql_execute
Select
TOTALSIZE
AS IOBUFFERCACHE_kB,
round(TOTALSIZE/8,0)
AS TOTALSIZE_Pages
DATACACHEUSEDSize
AS DATACACHE_kB,
round( DATACACHEUSEDSize/8)
AS DATACACHE_Pages,
CONVERTERUSEDSize
AS CONVERTERCACHE_kB,
round(CONVERTERUSEDSize/8)
AS CONVERTERUSEDSize_Pages,
(TOTALSIZE-DATACACHEUSEDSize-CONVERTERUSEDSize)
AS MISC
round((TOTALSIZE-DATACACHEUSEDSize-
CONVERTERUSEDSize)/8,4)
AS MISC_Pages
From IOBUFFERCACHES
```

Beispielhafte Ausgabe

Eine beispielhafte Ausgabe sehen Sie Abbildung 5.13, in der die einzelnen selektierten Werte nacheinander aufgelistet werden. Dies ist

natürlich nicht sehr übersichtlich, und Sie müssen die Werte entsprechend zu interpretieren wissen. Dazu empfehlen wir Ihnen, die Werte regelmäßig zu protokollieren, um Analysen darüber zu erstellen und Engpässe frühzeitig erkennen und beseitigen zu können.

```
/sapdb/programs/bin/dbmcli on g77db1 : G77>sql_execute Select TOTALSIZE, round(TOTALSIZE/8,0), DATACACHEUSEDSize,round( DATACACHEUSEDSize/8), CONVERTERUSEDSize,round( CONVERTERUSEDSize/8), (TOTALSIZE-DATACACHEUSEDSize-CONVERTERUSEDSize) from IOBUFFERCACHES
OK
END
3145728;393216;2949920;368740;184848;23106;10960
---
/sapdb/programs/bin/dbmcli on g77db1 : G77>
```

Abbildung 5.13 Ergebnis der SQL-Abfrage nach den Größen von Data Cache und Converter Cache

Weitere Caches im dbmcli auslesen

Neben den bisher besprochenen Caches existieren noch weitere Caches, deren Größen direkt konfigurierbar sind. Die Größen können bequem mit den folgenden Kommandos aus den Datenbankparametern ausgelesen werden. Gehen Sie dazu wie folgt vor:

Größen konfigurieren

1. Verbinden Sie sich zur Datenbank:

```
/opt/sdb/programs/bin/dbmcli -d MAXDB -n <host> -u
<user>,<passwort>
```

2. Führen Sie die folgenden Befehle aus, die Ihnen die aktuellen Größen der Caches ausgeben:

```
param_directget CAT_CACHE_SUPPLY
param_directget SEQUENCE_CACHE
```

```
/sapdb/programs/bin/dbmcli on g77db1 : G77>param_directget CAT_CACHE_SUPPLY
OK
CAT_CACHE_SUPPLY      10000
---
/sapdb/programs/bin/dbmcli on g77db1 : G77>param_directget SEQUENCE_CACHE
OK
SEQUENCE_CACHE      1
---
/sapdb/programs/bin/dbmcli on g77db1 : G77>
```

Abbildung 5.14 Auslesen der restlichen Größen aus den Datenbankparametern

Cache-Trefferraten im dbmcli auslesen

Wesentlich komfortabler gestaltet sich das Auslesen der Trefferraten der verschiedenen Caches. Hierzu benötigen Sie wieder SQL, da sich

diese Daten im Betrieb dynamisch ändern und so von der Datenbank wieder in Tabellen bereitgestellt werden. Der leichteren Auswertung dienen auch noch Beschreibungen der einzelnen Werte. In der Spalte DESCRIPTION wird eine kurze Beschreibung des jeweiligen Wertes dargestellt.

Trefferraten im dbmcli anzeigen

1. Verbinden Sie sich zur Datenbank:

```
/opt/sdb/programs/bin/dbmcli -d MAXDB -u control,control
```

2. Führen Sie den folgenden Befehl aus, der Ihnen die aktuelle Größe des Caches ausgibt:

```
sql_execute select * from monitor_caches
```

In Abbildung 5.15 sehen Sie das Ergebnis dieser Abfrage. Hierbei ist im Gegensatz zu den vorigen Statements keine zusätzliche Rechenarbeit nötig, da die Trefferrate bereits aus dem Verhältnis der gesamten Zugriffen zu den erfolgreichen Zugriffen errechnet und in der Systemtabelle `monitor_caches` abgespeichert wurde. An den Werten der OMS Caches ist zu sehen, dass es sich bei diesem Beispiel nicht um eine `liveCache`-Instanz handelt: So ist z.B. die Größe des OMS Caches gleich Null.

```

/sapdb/programs/bin/dbmcli on g77db1 : G77>sql_execute select *
from monitor_caches
OK
END
'Data cache accesses':6120894020
'Data cache accesses successful':6088745061
'Data cache accesses unsuccessful':32148959
'Data cache hit rate (%)':99
'Catalog cache accesses':1725497670
'Catalog cache accesses successful':1715057754
'Catalog cache accesses unsuccessful':10441125
'Catalog cache hit rate (%)':99
'Sequence cache accesses':0
'Sequence cache accesses successful':0
'Sequence cache accesses unsuccessful':0
'Sequence cache hit rate (%)':0
'Data cache sqlpage accesses':6039463266
'Data cache sqlpage accesses successful':6007314307
'Data cache sqlpage accesses unsuccessful':32148959
'Data cache sqlpage hit rate (%)':99
'Data cache OMS accesses':0
'Data cache OMS accesses successful':0
'Data cache OMS accesses unsuccessful':0
'Data cache OMS hit rate (%)':0
'Data cache OMS log accesses':0
'Data cache OMS log accesses successful':0
'Data cache OMS log accesses unsuccessful':0
'Data cache OMS log hit rate (%)':0
'Data history/undo accesses':81430754
'Data history/undo accesses successful':81430754
'Data history/undo accesses unsuccessful':0
'Data history/undo hit rate (%)':100
'Data cache no of SQL data pages':368590
'Data cache no of OMS data pages':0
'Data cache no of history/undo pages':0
'Data cache no of unloaded version pages':0
---
/sapdb/programs/bin/dbmcli on g77db1 : G77>

```

Abbildung 5.15 Trefferraten der Caches aus der Tabelle `monitor_caches`

5.4.5 Statistiken der kritischen Regionen

Die Caches sind in verschiedene Zugriffsbereiche, sogenannte *kritische Regionen*, unterteilt, um konkurrierende Zugriffe zu beschleunigen, die mit Sperren auf Datenbereichen arbeiten. Im Folgenden wollen wir Ihnen daher zeigen, wo Sie die kritischen Abschnitte in den wichtigsten Tools und Transaktionen erkennen.

Kritische Regionen erkennen

Kritische Regionen in der DB50

In der Transaktion DB50 können Sie sich die kritischen Abschnitte als Tabelle anzeigen lassen. Eine beispielhafte Ausgabe der Regionen sehen Sie in Abbildung 5.16.

Kritische Abschnitte (Regions)					
ID	Name	Kollisionsrate	Kollisionen	Wartesituationen	Zugriffe
49	CONV47	0,00	1.108	8	25.134.930
50	CONV48	0,00	622	5	25.137.306
51	DATA CACH	0,00	0	0	6.803
52	DCOM	0,00	0	0	0
53	DIAG CACH	4,09	233	1	5.695
54	FBM	0,07	222.612	1.137	315.120.916
55	GARBAGE	0,00	0	0	95.736
56	LOCKINIT	0,02	2.277	1	14.614.721
57	LOCKPOOL	0,04	6.597	0	17.663.889
58	LOCKREQ	0,09	10.731	0	11.721.874
59	LOCKWAIT	0,05	1.647	0	3.386.452
60	SAVEPOIN	0,00	0	0	12.222
61	LOGQUE1	1,35	2.518.118	1.848	186.247.596
62	LOGQUE2	0,00	0	0	2.247
63	LOGQUE3	0,00	0	0	2.247
64	LOGQUE4	0,00	0	0	2.247
65	LOGINFO	0,00	0	0	69.370
66	LVC_EM CY	0,00	0	0	0
67	MONITOR	467,83	97.462	892	20.833
68	SERVER	0,00	54	0	3.244.150
69	SRV TASK	4,79	4.913	42	102.533

Abbildung 5.16 Statistiken der kritischen Regionen in DB50

Um zu einer Übersicht zu gelangen, wie in der Abbildung 5.16 dargestellt, gehen Sie einfach wie folgt vor:

Kritische Regionen anzeigen

1. Loggen Sie sich auf dem SAP-System ein.
2. Starten Sie im SAP-System die Transaktion DB50.
3. Die Übersicht der Regionen erreichen Sie per **AKTUELLER STATUS • KRITISCHE REGIONEN**.

Wenn Sie in der Übersicht eine zu hohe Kollisionsrate entdecken, sollten Sie die entsprechenden Gegenmaßnahmen ergreifen, wie beispielsweise eine Erhöhung der Cache-Größe.

Kritische Regionen per dbmcli

Wie die Daten zu den Cache-Größen so sind auch die Daten zu den Zugriffstatistiken auf die kritischen Regionen nicht statisch, sondern werden fortwährend von der MaxDB protokolliert und verdichtet in der Tabelle `REGION_STATISTICS` hinterlegt.

Um sich die Daten der Regionen per Kommandozeile anzeigen zu lassen, gehen Sie wie folgt vor:

1. Verbinden Sie sich zur Datenbank:

```
/opt/sdb/programs/bin/dbmcli -d MAXDB -u control,control
```

2. Führen Sie den folgenden Befehl aus, der Ihnen die aktuelle Größe der Caches ausgibt:

```
dbmcli> sql_execute
select
    REGIONID
    AS ID,
    REGIONNAME
    AS Name,
    round((COLLISIONCOUNT*100)/ACCESSCOUNT,2)
    AS Kollisionsrate,
    WAITCOUNT
    AS Wartesituationen,
    ACCESSCOUNT
    AS Zugriffe
from REGIONSTATISTICS
where ACCESSCOUNT > 0
```

**Division durch Null
verhindert**

In diesem Beispiel wurden durch die `Where`-Klausel alle Werte vermieden, die zu einer Division durch Null geführt hätten. Dies stellt aber keine Beeinträchtigung des Informationsgehaltes dar, da durch den Wert der Spalte `ACCESSCOUNT` dividiert wird. Ist dieser Null, so wurde diejenige kritische Region nicht betreten und kann somit auch keine Wartezeiten verursacht haben. Die Abbildung 5.17 zeigt die Ausgabe dieses SQL-Statements.

```

49; 'CONV47'; 0.00; 8; 25135993
50; 'CONV48'; 0.00; 5; 25138355
51; 'DATA CACH'; 0.00; 0; 6803
53; 'DIAG CACH'; 3.86; 1; 6042
54; 'FBM'; 0.07; 1137; 315137499
55; 'GARBAGE'; 0.00; 0; 95904
56; 'LOCKINIT'; 0.02; 1; 14629055
57; 'LOCKPOOL'; 0.04; 0; 17679329
58; 'LOCKREQ'; 0.09; 0; 11733687
59; 'LOCKWAIT'; 0.05; 0; 3388906
60; 'SAVEPOINT'; 0.00; 0; 12233
61; 'LOGQUE1'; 1.35; 1848; 186347507
62; 'LOGQUE2'; 0.00; 0; 2249
63; 'LOGQUE3'; 0.00; 0; 2249
64; 'LOGQUE4'; 0.00; 0; 2249
65; 'LOGINFO'; 0.00; 0; 69399
67; 'MONITOR'; 467.83; 892; 20833
68; 'SERVER'; 0.00; 0; 3247021
69; 'SRVTASK'; 4.79; 42; 102541
70; 'SURROGAT'; 0.01; 0; 17471859
71; 'CONVINDX'; 0.00; 0; 212884
72; 'KNLWAIT'; 0.00; 0; 17

```

Abbildung 5.17 Zugriffsstatistiken der kritischen Regionen

5.5 Analyse-Tools

Wurden alle Indizes korrekt eingerichtet und alle Caches ausreichend dimensioniert, so kann es im Verlaufe der Benutzung der Datenbank, durch Wachstum der Daten und Veränderungen im Nutzungsverhalten dazu kommen, dass die Performance nachlässt. Ein grundlegendes Verständnis der Datenbankfunktionalitäten und ihrer internen Abläufe ist eine wichtige Voraussetzung für das Aufspüren von Performance-Engpässen. MaxDB bietet eine Reihe von Analyse-Tools, die in der Auslieferung enthalten sind und im Zuge der Installation zusammen mit dem Datenbanksystem installiert werden. Die folgenden Abschnitte werden den Teil der mitgelieferten Tools beschreiben, die für das Aufspüren von Performance-Engpässen vorgesehen sind. In Abschnitt 5.6.1, »Performance SAP NetWeaver AS«, werden wir Ihnen Möglichkeiten vorstellen, wie Sie die in diesem Kapitel vorgestellten Tools innerhalb von SAP anwenden und deren Ergebnisse visualisieren können.

In diesem Abschnitt stellen wir Ihnen die wichtigsten Analyse-Tools in der Reihenfolge vor, wie sie häufig in der Praxis benutzt werden. Der *Database Analyzer* dient der ständigen Überwachung der Vitalwerte der Datenbank. Er bietet Ihnen die Möglichkeit, sich unnormale Zustände melden zu lassen, um dann in den entsprechenden Informationsdateien eine erste Problemanalyse durchzuführen.

Einsatzstrategie für die Analyse-Tools

Index

32-Bit 15

A

ABAP-Stack 100
absoluter Pfad 91
ACTION 139
ADA_SQLDBC 118
ADABAS 12
adm 59
After-Image 75
Analyse-Tool 265
appldiag 317
Applikationsebene 102
Applikationsserver 99
AS ABAP 100
AS Java 100
Asdev Thread 64
Ausführungskosten 242
Ausführungsplan 232
automatisches Log Backup 153, 193

B

B*-Baum 73, 234, 236
Backup 180
 Check last backup 203
 Dauer 190
 durchführen per dbmcli 191
 durchführen per DBMGUI 189
 History 182, 199, 212
 inkrementelles 189
 Konzepte 181
 Medium/Template prüfen 205
 Sicherungsarten 182
 Sicherungsstrategie 183
 Status abfragen 191
 Status der Überprüfung 204
 Template 184
 Überprüfung per dbmcli 204
 Überprüfung per DBMGUI 202
Backup-Medium 184
 anlegen per dbmcli 187
 anlegen per DBMGUI 185
 Eigenschaften 186

löschen 188
 löschen per dbmcli 188
 löschen per DBMGUI 188
 paralleles Medium 186
 Typen von Medien 184
Bad Index 215
 beseitigen per dbmcli 217
 beseitigen per DBMGUI 216
 erkennen 215
Befehlsreferenz 39
Before-Image 75
Benutzer-Control 141
Benutzergruppe 57
Benutzerrechte 52, 53, 54
Benutzertyp 51
Betriebsmodus 37
Betriebssystem 14
Betriebssystembenutzer 58
Betriebszustand 93

C

Cache 71, 252, 253, 257
 Größe 78, 257
 Trefferrate per dbmcli 261
CacheMemorySize 156
Catalog Cache 77, 253, 255
Client-Server-Architektur 29, 102
Command 282
Command Monitor 283, 285, 307
 Konfiguration 285
Configuration Type 147
 Custom 147, 149
 Desktop PC/Laptop 147
 My Templates 147
Console Thread 63
Converter 71, 73
Converter Cache 255
Coordinator Thread 63

D

Data Area Extension 330
Data Cache 71, 254
Data Volume 79, 144

- anlegen* 152
 - anlegen eines dynamischen Data Volumes* 165
 - anpassen* 152
 - Eigenschaften* 163
 - hinzufügen per dbmcli* 164
 - hinzufügen per DBMGUI* 162
 - löschen* 169
 - Volume-Restriktion* 145
 - Data Warehouse 24
 - Database Analyzer 47, 266, 298
 - Dateipfad* 269
 - Konfigurationsdatei* 269
 - Log-Datei* 276
 - Ressourcenverbrauch* 267
 - starten per dbmcli* 267
 - starten per DBMGUI* 266
 - starten per Kommandozeile* 267
 - Database Manager
 - CLI* 37, 93
 - GUI* 33, 93
 - Operator* 51, 150
 - Database Monitor 124, 125
 - Database Studio 30, 32, 93
 - Database System Administrator 153, 223
 - Datenbank
 - aktivieren* 157
 - anlegen auf Kommandozeile* 155
 - anlegen per Skript* 158
 - Assistent* 114
 - Ebene* 102
 - erzeugen per GUI* 146
 - Instanz* 35, 61, 62, 94
 - konfigurieren* 162
 - Konsole* 118
 - löschen per dbmcli* 224
 - löschen per Installation Manager* 223
 - Operatoren* 52, 53
 - Parameter* 94
 - planen* 144
 - Trace* 118
 - Datenbankoptimierer 243
 - Aktualisierung Optimizer-Statistiken* 245
 - Cost-Based Optimizer* 243
 - Größe der Probe* 247
 - Optimizer-Statistiken* 244
 - Rule-Based Optimizer* 243
 - Zugriffsstrategie* 251
 - Datenexport 48
 - Datenimport 49
 - Datensatzsperrung 333
 - Datentransport 48
 - DB50 258, 263, 298
 - DBA 55
 - DBA History 121
 - DBA-Einplanungskalender 118, 122
 - DBM Operator 52
 - dbm.ebl 321, 335
 - dbm.ebp 321, 335
 - dbm.knl 320
 - dbm.prt 317
 - dbm.utl 319
 - dbmcli 38
 - DBMGETF 49
 - DBMGui 33
 - DB-Zeit 293, 296
 - Dependent Program Path 90, 91, 133
 - Dev-Thread 63
 - Dev-Trace 314
 - Diagnosedatei 313
 - Dispatcher 101
 - Dokumentation 17
 - Drei-Schichten-Architektur 103
 - Drill-down 24
- ## E
-
- Einsatzgebiet 16
 - Ein-Schicht-Architektur 102
 - Entstehungsgeschichte 11
 - Equal Condition for Index Column 251
 - Event 69
 - Exclusive Lock 88
 - EXPLAIN 290
- ## F
-
- Festplatte 62
 - FILE 85
 - File Directory 71, 74
 - Full Table Scan 251
- ## G
-
- Garbage Collector 28
 - GETDBROOT 50

H

Hauptspeicher 62

Hot Standby 37

I

I/O Buffer Cache 26, 71, 253

I/O-Worker-Thread 64

Independent Data Path 90, 91, 132

Independent Program Path 90, 91, 132,
138, 163

Indizes 233

Ausführungskosten 242*B*-Baum* 233*invertierte Liste* 240*LONG-Werte in B*-Bäumen* 238*Primärschlüssel* 235*Sekundärschlüssel* 235

Installation

Fehlersuche 138*im Dialog* 128*im Hintergrund* 134*Manager* 135, 222*Phase* 131*Protokolldatei* 138*Type* 137

Installationsprofil 130

INSTALLER_INFO 138

Instanztyp 23

Interface 107

invertierte Liste 240, 241

IPC (Inter-Process Communication) 111

Isolationslevel 89

J

Java-Stack 100

JDBC (Java Database Connectivity) 108

JDBC-Interface 42

Joins 252

K

Kernel 61

Thread 62*Trace* 323*Variante* 94

knldiag 318

knldiag.err 319

knldump 324

KnlMsg 318

KnlMsgArchive 319

knltrace 323

Komponentengruppe 128

Konfigurationsdatei 96, 269

Konfigurationstyp My Templates 154

Konsistenzprüfung 218

Datenbankstruktur 219*Datenbankstruktur per dbmcli prüfen*
221

kooperatives Multitasking 66

kritische Region 263

L

Lastanalyse 296

LINK 85

Linux 14, 31

liveCache 25, 255

Lizenz 12

Load Balancing 70

Loader 47

Log Backup 83, 192

automatisches 193*durchführen per dbmcli* 196*durchführen per DBMGUI* 194

Log Full 84

Log I/O Queue 253, 256

Log Queue 75

Log Volume 81

anlegen 152*anlegen per dbmcli* 167*anlegen per DBMGUI* 166*anpassen* 152*Eigenschaften* 166*Overwrite-Modus* 158*Spiegel reintegrieren* 213*spiegeln* 171

Log Writer 69

Log-Datei 276

Log-Modus

Konfiguration 173*Overwrite-Modus* 173*Redo-Log-Management* 174

Log-Partition 81

Log-Segment 84

Log-Sicherung 123

M

MaxCPUs 66
 Microsoft Windows 15
 Mirroring 83
 Monitor 273
 MSG 139

N

.NET Wrapper 109

O

Object Identifier 27
 ODBC 107
 Offline-Modus 142
 OLAP (Online Analytical Processing) 17,
 23, 25
 OLAP-Cubes 24
 OLTP (Online Transaction Processing)
 16, 23
 OMS (Object Management System) 26
 OMS Cache 255
 OMS Heap 28
 Optimierertypen 243
 Optimistic Lock 88
 Optimizer-Statistik 244, 245, 246, 252
 Overwrite-Modus 153

P

Page 27
 Page Chain 27
 Pager 69
 Parallelisierung 68
 Parameter
 _IOPROCS_PER_DEV 144
 ändern 176
 ändern per dbmcli 178
 ändern per DBMGUI 177
 auf andere Datenbank kopieren 179
 committen 156
 Gruppe 94
 initialisieren 156
 kopieren 179
 Parameterkategorie 177
 Parameter-Session starten 156
 Sitzung 96

Parameterinitialisierung 150
 Copy Parameters from Existing Database
 151
 Initialize Parameters with Default Values
 151
 Restore Parameters from a Backup 151
 Use Current Parameters 151

Performance 232

Perl 110

pgm/kernel 142

PHP 110

Pointer 27

Port 111

Positionsindex 27

Präsentationsebene 101

Preparing-Phase 131

Primary Key 249

Problemsituation

Data-Full-Situation 327, 330

Hardwarefehler 336

Log-Full-Situation 327

Systemabsturz 331

Systemblockade 332

Verbindungsprobleme 326

Python 110

R

RAID 80, 145

Range Condition for Key Column 251

RAW 85

RAW Device 146

Recovery 206

durchführen per dbmcli 212

durchführen per DBMGUI 209

mit Initialisierung 209, 212

ohne Initialisierung 212

Strategie 207

Typ 206

Requestor Thread 63

Resource Monitor 278, 279, 280, 302,
 303

ROLAP (Relational OLAP) 25

Rollenkonzept 57

Roll-up 24

root 59

Root Page 235

rtedump 322

RUNDIRECTORY 180

S

- SAP 105, 107
- SAP CCMS 114, 124
- SAP Content Server 17
- SAP DB 12
- SAP Developer Network 18
- SAP NetWeaver AS 293, 297
- SAP-Architektur 99
- SAPCAR 128
- SAPInst 80, 160
 - Fehlerfall* 161
 - Log-Datei* 161
 - Log-Datei für MaxDB-Installation* 161
 - Phasen* 160
- SAP-Landschaft 112
- SAProuter 112
- SAP-Standardbenutzer 57, 105
- Savepoint 72, 80, 86
- Schattenspeicherkonzept 72
- Schnittstelle 15
- sdb 59
- sdba 59
- SDBINST 127, 128, 129, 134
- SDBREGVIEW 49
- SDBSETUP 127, 135, 226
- SDBUPD 139, 140, 141, 142
- Selektivität 287
- Sequence Cache 78
- Server Task 68
- Server-Landschaften 32
- Server-Software
 - Deinstallation* 226
 - Deinstallation per sdbunsint* 228
- Service Session 204
- Servlet-Container 42
- Shared Lock 88
- Shared SQL Cache 77
- Sicherheitsaspekte 60
- Slice and Dice 24
- Snapshot 197
 - anlegen per dbmcli* 198
 - anlegen per DBMGUI* 198
 - Funktionsweise* 197
 - löschen per dbmcli* 201
 - löschen per DBMGUI* 201
 - Revert per dbmcli* 200
 - Revert per DBMGUI* 200
- Softwarekomponentengruppe 129
- Sperr-Eskalation 89
- Sperrliste 89
- SQL CLI 40
- SQL Editor 32
- SQL Explain 289
- SQL Studio 39, 118
- sql30 111
- sql6 111
- SQL-Benutzer 55
- SQLDBC 108
- SQLDBC-Trace 119, 315
- SQL-Modus 15
- SQL-Schnittstelle 113
- SQL-Trace 314
- STDIN 138
- STDOUT 138
- Sternschema 24
- Striping 81
- Suchkriterium 242
- Support-Gruppen 60
- SYS 138
- SYSDBA-User 51, 55
- SYSMONITOR 283
- Systemtabelle 142, 157, 175
 - laden* 176
 - Systemtabellenkategorie* 175

T

- Table Editor 32
- TCP Port 36
- Template 148
- Timer 69
- Timer Thread 63, 70
- Tomcat 43
- Trace Writer 69
- Trace-Datei 120
- Transaktion 114
 - CCMS* 337
 - DB12* 120
 - DB13* 121
 - DB50* 114, 258, 297, 302, 307, 310
 - DBCO* 105, 106
 - RZ20* 124
 - ST03N* 293, 296
- Transaktionsprofil 294
- Tutorial Data 153

U

Uninstallation Summary 227
Unix 14
Update 140
Upgrade 142
 Dauer 143
 versus Update 139
User Kernel Thread 66, 68
User Task 67, 104
Utility 69
Utility Session 196, 212
 öffnen 191

V

Versionsbezeichnung 15
Verzeichnisstruktur 90, 92
View 270
Visual Query Editor 32
Vwait 334

W

Watchdog Process 65
Web Database Manager 43
Web DBM 43
Web SQL 41
Work-Prozess 104

X

X_CONS 43, 65
X_PING 50
XINSTINFO 50
X-Server 29, 111, 140, 141
X-Server-Protokoll 316
XUSER 45

Z

zentrale Benutzerverwaltung 60
Zwei-Schichten-Architektur 102
zyklisches Schreiben 82