HANSER

Leseprobe

Andrea Held, Mirko Hotzy, Lutz Fröhlich, Marek Adar, Christian Antognini, Konrad Häfeli, Daniel Steiger, Sven Vetter, Peter Welker

Der Oracle-DBA

Handbuch für die Administration der Oracle Database 11g R2

ISBN: 978-3-446-42081-6

Weitere Informationen oder Bestellungen unter

http://www.hanser.de/978-3-446-42081-6

sowie im Buchhandel.

© Carl Hanser Verlag, München

4

4 Speicherplatzverwaltung

In diesem Kapitel zeigen wir Ihnen:

- wie die grundlegenden Speicherstrukturen einer Oracle Datenbank aufgebaut sind;
- wie damit die unterschiedlichen Segmenttypen verwaltet werden;
- die Vor- und Nachteile der verschiedenen Verwaltungsmethoden;
- welche Reorganisationsmethoden Oracle zur Verfügung stellt.

In einer Oracle-Datenbank sind Objekte, die Daten beinhalten (z.B. Tabellen und Indizes), nicht direkt in Files auf Betriebssystemebene gespeichert, sondern logischen Strukturen zugeordnet, die ihrerseits in physischen Strukturen abgelegt sind. Ziel dieser Architektur ist die klare Trennung des logischen Designs von der physischen Implementierung. Abbildung 4.1 zeigt eine Übersicht der Speicherstrukturen und deren Beziehung zueinander.

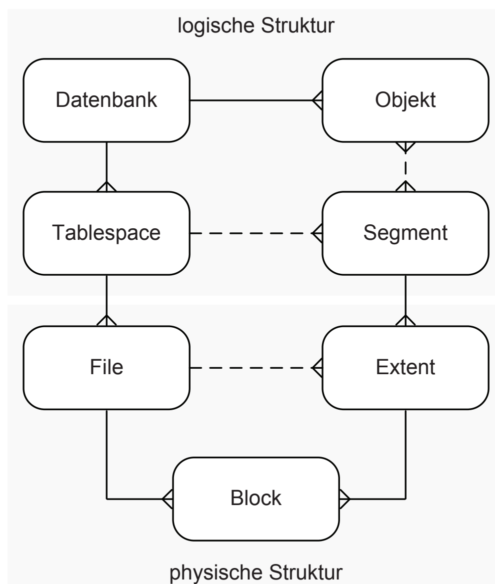

Abbildung 4.1
Übersicht der Datenbank-Speicherstrukturen

Eine Datenbank besteht aus mehreren Tablespaces. Datenbank und Tablespaces sind lediglich logische Strukturen, die Objekte bzw. Segmente beinhalten. Man kann sie auch als eine Art Container betrachten. Aus Sicht der Speicherung unterscheidet man vier Objekttypen:

- Objekte, die keine Daten und demzufolge auch keine Segmente beinhalten. Eine View ist beispielsweise eine logische Struktur, die nur im Data Dictionary definiert ist
- Objekte, die aus einem einzigen Segment bestehen, z.B. eine nichtpartitionierte Heap-Tabelle
- Objekte, die aus mehreren Segmenten bestehen. Eine partitionierte Tabelle hat z.B. ein Segmente pro Partition
- Objekte, die ein Segment mit einem anderen Objekt teilen, z.B. Cluster-Tabellen

Jeder Tablespace besteht aus einem oder mehreren Files, die aus einer bestimmten Anzahl Blöcke bestehen. Blöcke sind die kleinsten Strukturen, die die Datenbank speichertechnisch verwaltet. Über mehrere Files pro Tablespace verhindert man Einschränkungen, die durch die Database-Engine, das Betriebssystem oder die Hardware bezüglich der maximalen Filegröße entstehen können. Jedes File ist in Einheiten von zusammenhängenden Blöcken, den sogenannten „Extents“, unterteilt. Jedes Extent ist genau einem einzigen Segment zugeordnet.

In den folgenden Abschnitten beschreiben wir die Möglichkeiten der Datenbankfile-Speicherung und wie man die erwähnten logischen und physischen Speicherstrukturen verwaltet.

4.1 Datenbank-Speicheroptionen

Oracle unterstützt die unterschiedlichsten Speicherarchitekturen. Im einfachsten Fall werden die Disks direkt an den Datenbankserver angeschlossen. Man spricht dann von Direct Attached Storage (DAS). Weitaus häufiger wird auf die Disks in den heutigen Rechenzentren jedoch über ein Netzwerk zugegriffen. Dabei kommen hauptsächlich zwei Architekturen zum Einsatz: Storage Area Networks (SAN) und Network Attached Storage (NAS). Ein SAN verfügt über einen Management Layer und über ein spezielles Netzwerk, das die physische Verbindung zu den Disks sicherstellt. Die NAS-Architektur verwendet spezielle Fileserver, die für Speicherlösungen über ein lokales Netzwerk (LAN) geeignet sind. Die DAS- und SAN-Architekturen erlauben Block-I/O-Zugriff, während NAS-Server den Zugriff über File-I/O ermöglichen.

Oracle nutzt für die Verwaltung der Datenbankfiles die betriebssystemseitigen Möglichkeiten. Dies hat aus Sicht der Datenbankadministration den Vorteil, dass die eingesetzte Speicherarchitektur die Fileverwaltung auf Datenbankebene nicht beeinflusst. Aus diesem Grund gehen wir an dieser Stelle nicht tiefer auf die erwähnten Architekturen ein und konzentrieren uns auf die Anforderungen, die ein Speichersubsystem aus Datenbanksicht erfüllen muss. Wir betrachten insbesondere, wie diese Anforderungen mit den drei grundlegenden Methoden erfüllt werden, die Oracle für die Speicherung von Datenbankfiles

verwendet: Filesysteme, Raw-Devices und Automatic Storage Management (ASM). Abgerundet wird dieses Kapitel mit Empfehlungen für die Auswahl von Datenbank-Speichersystemen.

4.1.1 Eigenschaften eines Speichersystems

Ein zuverlässiges Speichersystem ist für die dauerhafte Speicherung von Datenbankfiles unabdingbar. Die Hauptanforderungen an ein Speichersystem sind:

- Verwaltung: Einfache und flexible Verwaltung der Disks
- Verfügbarkeit: Schutz gegen Hard- und Softwarefehler
- Performance: Erforderlichen Durchsatz und Anzahl I/O-Operationen pro Sekunde
- Zugriff: Gemeinsamer Zugriff auf die gleichen Disks von mehreren Servern aus in einer Clusterumgebung

Bestimmte Anforderungen widersprechen einander jedoch. So existiert beispielsweise oft ein Konflikt zwischen einfacher, flexibler Verwaltung und hoher Performance. Im Falle des gleichzeitigen Zugriffs ist die Anforderung bei bestimmten Systemen gar nicht relevant. Es ist deshalb für die Wahl eines Speichersystems entscheidend, die Features entsprechend zu gewichten.

4.1.1.1 Verwaltung

Operationen wie das Hinzufügen, Entfernen oder Ersetzen einer Disk sollten einfach und unterbrechungsfrei ohne Stoppen der Datenbankinstanz möglich sein, im Idealfall sogar ohne Auswirkungen auf die Datenbankinstanz – auch wenn auf den betroffenen Disks bereits Teile der Datenbank gespeichert sind. Wenn eine solche Operation durchgeführt wird, ist es zudem essenziell, dass die bereits gespeicherten Daten bei Bedarf automatisch neu verteilt werden. Solche Funktionen können nur dann zur Verfügung stehen, wenn eine Virtualisierungsschicht, beispielsweise ein Logical Volume Manager (LVM), zwischen Datenbankinstanz und physischen Disks zum Einsatz kommt. Wenn solche Features nicht unterstützt sind, sind die betroffenen Tablespaces manuell offline zu nehmen und/oder die Daten manuell zu verschieben.

Ein Speichersystem sollte Operationen wie Datenbankfiles verschieben oder kopieren ebenfalls unterstützen. Diese Operationen erfordern jedoch ein Wartungsfenster, weil die betroffenen Tablespaces offline, oder in den Backupmodus genommen werden müssen – ansonsten ist die Konsistenz der Datenbankfiles nicht mehr gewährleistet.

Weil Oracle das Vergrößern und Verkleinern (Resize) von Datenbankfiles unterstützt, sollte auch das Speichersystem dazu in der Lage sein.

4.1.1.2 Verfügbarkeit

Der Schutz gegenüber Hard- und Softwarefehlern sowie gegenüber menschlichen Fehlern ist für Datenbanken von höchster Bedeutung: den Verlust einer abgeschlossenen Trans-

aktion gilt es unter allen Umständen zu verhindern. Einerseits verfügt Oracle über diverse Features, die – sofern korrekt angewendet – sicherstellen, dass keine abgeschlossene Transaktion verloren geht. Der Einsatz dieser Möglichkeiten liegt meistens beim Applikationsverantwortlichen, wobei es vielfach aus Kostengründen nicht möglich ist, das System gegen alle möglichen Fehler zu schützen.

Andererseits verfügen die Speichersubsysteme über bewährte Ausfallschutzoptionen. Die bekannteste Methode, um Daten gegen Hardware-Ausfall zu schützen, ist das Duplizieren von Hardware-Komponenten wie Disks und Interfaces. Üblich ist beispielsweise die Spiegelung der Daten auf zwei unabhängige Disks, auch bekannt unter dem Begriff „RAID 1".

Aus Performance-Gründen empfehlen wir, solche Spiegelungen auf Hardware-Ebene umzusetzen. In einer DAS-Umgebung sind dazu Controller erforderlich, welche die Spiegelung unterstützen. Spiegelung wird heutzutage von jedem SAN oder NAS unterstützt. Trotz Spiegelung auf Hardware-Ebene macht die zusätzliche Spiegelung auf Software-Ebene, beispielsweise für die Control-Files, die Redolog-Files und die archivierten Log-Files Sinn (siehe Kapitel 2, „Architektur und Administration"). Die meisten Datenbanken nutzen die Möglichkeiten der Spiegelung und es gibt keinen Grund, auf diese zu verzichten. Weitere Möglichkeiten der Spiegelung stellen ASM oder ein LVM zur Verfügung.

Eine Problematik der Hardware-Spiegelung ist der doppelte Platzbedarf. Devices, die Datenredundanz auf Basis von Parity-Information (z.B. RAID 5 und RAID 6) zur Verfügung stellen, verringern den Overhead in Bezug auf den Diskplatz stark. Anstelle der vollständigen Replizierung der Daten speichern diese Technologien lediglich die Parity-Information, die für die Rekonstruktion von fehlenden Daten notwendig ist. Der Nachteil dieser Methode liegt in einer allgemein schlechteren Schreib-Performance. Wir empfehlen daher deren Einsatz für Oracle-Datenbanken nicht.

Praxistipp

Für die optimale Performance sollte man parity-basierte Datenredundanz-Technologien wie RAID 5 und RAID 6 nicht einsetzen.

4.1.1.3 Performance

Für die optimale Performance ist die Wahl der Hardware von höchster Bedeutung. Es ist deshalb sehr wichtig, diesen Aspekt bei den Auswahlkriterien zu berücksichtigen.

Praxistipp

Obwohl die Kapazität in Bits und Bytes ein Schlüsselkriterium für die Wahl eines Speichersystems ist, sind die erforderlichen I/O-Operationen pro Sekunde (IOPS) und deren Größe nicht weniger wichtig. Im Gegenteil: Ein Speichersystem sollte nicht ausschließlich aufgrund seiner Kapazität ausgewählt werden.

Dazu ein Beispiel: Für eine OLTP-Datenbank von 1,4 TB Größe sind die notwendigen Disks (Typ und Anzahl) zu evaluieren, damit eine Spitzenlast von 1000 IOPS à 8 KB be-

wältigt werden kann. Die Daten sollten zudem gespiegelt sein und es stehen zwei Disktypen zu Auswahl:

- High Performance SAS-Disks mit insgesamt 600 GB Kapazität und einem Durchsatz von 175 IOPS à 8 KB
- High Capacity SATA-Disks mit insgesamt 2 TB Kapazität und einem Durchsatz von 75 IOPS à 8 KB

Wenn wir nur die Kapazität in Betracht ziehen, sind die Anforderungen mit 2 SATA-Disks (2*1.4/2.0) oder 6 SAS-Disks (2*1.4/0.6) zu erfüllen. Wenn wir jedoch die IOPS berücksichtigen, müssen mehr Disks eingesetzt werden. Um 1000 IOPS zu erreichen, sind mindestens 28 SATA-Disks (2*1000/75), oder 12 SAS-Disks (2*1000/175) erforderlich. Diese Betrachtung zeigt deutlich auf, dass bei der Auswahl des Speichermediums nicht nur die Kapazität, sondern auch die Performance-Anforderungen wichtig sind.

Die Auswahl des richtigen Disktyps und der richtigen Anzahl Disks ist schon einmal ein guter Anfang. Aber Vorsicht: Die erforderliche Spitzenlast ist nur dann gewährleistet, wenn die Disks korrekt und in geeigneter Art und Weise mit dem Server verbunden sind. In einer DAS-Umgebung müssen genügend Controller verfügbar sein und in einer SAN/NAS-Umgebung muss die Netzwerkinfrastruktur (HBA[1], Netzwerkkarten und Switches) adäquat dimensioniert sein.

Ein wichtiger Aspekt für die optimale Nutzung der verfügbaren Hardware ist die gleichmäßige Verteilung der Last über alle Komponenten.

Praxistipp

Die manuelle Verteilung der Speicher-Operationen, beispielsweise durch die Zuordnung von einzelnen Disks zu spezifische Tablespaces, ist in der Regel wenig sinnvoll. Für die Maximierung der Performance empfehlen wir Striping (RAID 0) zu verwenden und die automatische Verteilung der Speicher-Operationen über die verfügbaren Komponenten dem Speicher-Subsystem zu überlassen.

Die einzige Situation in der eine manuelle Verteilung in Betracht kommt, besteht dann, wenn Disks mit unterschiedlicher Performance zur Verfügung stehen. Es ist beispielsweise sinnvoll, die schnellsten Disks für die Redolog-Files und für häufig benutzte Datenfiles einzusetzen, während die Disks mit der größten Kapazität mehr für sporadisch genutzte Daten zum Einsatz kommen.

Für die optimale Performance empfehlen wir, Striping auf Hardware-Ebene zu implementieren. In einer DAS-Umgebung muss dazu ein Controller verwendet werden, der Striping unterstützt. Striping ist eine Funktionalität, die heutzutage von jedem SAN oder NAS unterstützt wird.

Wenn sowohl hohe Verfügbarkeit als auch hohe Performance erforderlich sind, dann sollte man Spiegelung und Striping kombinieren (RAID 1+0 oder RAID 10). Das heißt, die Disks werden paarweise gespiegelt und das Striping erfolgt über die beiden Disks. Vorsicht ist im umgekehrten Fall geboten. Erfolgt die Spiegelung über gestripte Disks (RAID 0+1), verliert man den Vorteil der höheren Verfügbarkeit.

---

[1] Host Bus Adapter (HBA): Mediator, vermittelt zwischen dem SAN und dem Datenbankserver

Der höchstmögliche Nutzen der verfügbaren Hardware hängt auch von der eingesetzten Software ab. Auf Software-Ebene sind daher zwei Dinge zu beachten: Erstens sollte das Betriebssystem I/O-Operationen bis zu einer Größe von mindestens 1 MB unterstützen. Zweitens sollte asynchrones I/O[2] nicht nur vom Betriebssystem unterstützt werden, sondern auch auf Datenbankebene über den Parameter `disk_asynch_io` eingeschaltet sein (Defaultwert ist `TRUE`). Der Parameter ist eine Art Hauptschalter, über den asynchrone I/O-Operationen für Datenbankfiles auf beliebigen Speichersystemen ein- oder ausgeschaltet werden.

4.1.1.4 Zugriff

In einer Single-Instanz-Umgebung ist es normalerweise nicht notwendig, den gleichzeitigen Zugriff auf das Speichersubsystem von mehreren Servern aus zu ermöglichen. Die einzige Ausnahme besteht in einer Failovercluster-Umgebung. Doch selbst in diesem Fall ist es besser, die erforderlichen Disks (beziehungsweise Devices) exklusiv auf jenem Clusterknoten zu mounten, auf dem die Instanz aktiv ist.

Im Gegensatz dazu müssen in einer Real-Application-Cluster-Umgebung die Datenbankfiles von allen Instanzen aus simultan zugreifbar sein – daran führt kein Weg vorbei.

4.1.2 Filesysteme

Datenbankfiles sind am häufigsten auf Filesystemen gespeichert. Der Grund dafür ist einfach: Filesysteme sind weit verbreitet und der Umgang damit entsprechend vertraut. Die Oracle Binaries werden auf einem Filesystem installiert und auch die Datenbankfiles lassen sich ohne zusätzliche Konfiguration darauf speichern.

Jedes Betriebssystem unterstützt diverse Filesysteme, aber nicht jedes Filesystem ist von Oracle unterstützt – beispielsweise ist der Support für Linux von der Distribution und der Version abhängig. Details dazu stehen in den folgenden My Oracle Support Notes :

- Linux: Supported and Recommended File Systems on Linux (236826.1)
- SuSE/Novell: Linux, Filesystem & I/O Type Supportability (414673.1)
- Enterprise Linux: Linux, Filesystem & I/O Type Supportability (279069.1)
- AIX: Direct I/O or concurrent I/O on AIX 5L (272520.1)

Auf Plattformen wie Linux, Solaris, AIX und HP-UX unterstützt Oracle auch Filesysteme von Drittanbietern, wie zum Beispiel die Veritas Storage Foundation von Symantec.

In den folgenden Abschnitten beschreiben wir die Eigenschaften von Filesystemen bezüglich Verwaltung, Verfügbarkeit, Performance und Zugriff.

[2] Asynchrones I/O, auch „non-blocking I/O“ genannt, ist eine sehr effiziente Form von Input/Output-Verarbeitung, die parallele I/O-Verarbeitung ermöglicht. So muss beispielsweise ein Schreibprozess nicht auf die Schreibbestätigung warten, um weiterzuarbeiten. Asynchrones I/O muss vom Betriebssystem unterstützt sein.

4.1.2.1 Verwaltung

Operationen wie das Hinzufügen, Entfernen oder Ersetzen einer Disk (inklusive Wiederverteilung der Daten) kann nur dann unterbrechungsfrei (ohne Stoppen der Datenbankinstanz) erfolgen, wenn ein Logical Volume Manager oder ein Netzwerk-Device, das solche Features unterstützt, eingesetzt sind.

Operationen wie Verschieben oder Kopieren von Datenbankfiles können – nach vorgängiger Benachrichtigung der Datenbankinstanz – mit den allgemein bekannten Betriebssystembefehlen (wie `cp` oder `mv` unter Linux/UNIX) erfolgen. Das folgende Beispiel, ausgeführt auf einem Linux-Server, zeigt wie ein Datenbankfile eines Tablespaces von einem Verzeichnis in ein anderes verschoben wird:

```
SQL> SELECT file_name
  2  FROM dba_data_files
  3  WHERE tablespace_name = 'TEST';

FILE_NAME
-------------------------------------
/u00/oradata/ora11g/test01ora11g.dbf

SQL> ALTER TABLESPACE test OFFLINE;

SQL> !mv /u00/oradata/ora11g/test01ora11g.dbf -
>        /u01/oradata/ora11g/test01ora11g.dbf

SQL> ALTER DATABASE
  2  RENAME FILE '/u01/oradata/ora11g/test01ora11g.dbf'
  3  TO '/u00/oradata/ora11g/test01ora11g.dbf';

SQL> ALTER TABLESPACE test ONLINE;
```

Auch das Vergrößern (respektive Verkleinern) eines Datenbankfiles wird von Oracle unterstützt. Die einzige Voraussetzung, welche natürlicherweise erfüllt sein muss, ist genügend freier Platz im entsprechenden Filesystem. In Abschnitt 4.2.2 und 4.2.3 zeigen wir je ein Beispiel für automatisches und manuelles Resizing.

4.1.2.2 Verfügbarkeit

Mit Ausnahme der Control-Files, der Redolog-Files und der archivierten Log-Files (die auch softwaremäßig gespiegelt werden können) kann die Verfügbarkeit der Files auf dem Filesystem nur durch den Einsatz von Spiegelung auf Hardware-Ebene, oder mithilfe eines Logical Volume Managers gewährleistet werden.

4.1.2.3 Performance

Die Datenbankblockgröße sollte gleich oder ein Vielfaches der Systemblockgröße sein. Ist sie kleiner, ist die Performance nicht optimal. Das heißt, es muss ein ganzer Filesystemblock von der Platte gelesen werden, wenn eine I/O-Operation durchgeführt wird.

Standardmäßig sind die I/O-Operationen von Oracle auf den meisten Filesystemen gepuffert. Das heißt, die Daten sind in einem betriebssystemseitigen Buffercache gespeichert, nachdem sie gelesen bzw. bevor sie geschrieben werden. Weil Oracle jedoch einen eigenen

Buffercache besitzt, führt eine solche Doppelpufferung im Allgemeinen zu unnötigem Overhead.

Praxistipp

Wir empfehlen, die Pufferung von I/O-Operationen zu verhindern. Dazu unterstützen die Filesysteme Direct-I/O, womit – wie der Name impliziert – der betriebssystemseitige Buffercache umgangen wird.

Der Parameter `filesystemio_options` bestimmt, wie die I/O-Operationen auf Datenbankfiles gegenüber dem Filesystem ausgeführt werden. Folgende Werte stehen zur Verfügung:

- `none`: Deaktiviert direkte und asynchrone I/O-Operationen
- `directIO`: Aktiviert nur direkte I/O-Operationen
- `asynch`: Aktiviert nur asynchrone I/O-Operationen
- `setall`: Aktiviert direkte und asynchrone I/O-Operationen

Der Parameter `filesystemio_options` ist für die Aktivierung von asynchronen I/O-Operationen nicht möglich, wenn der Parameter `disk_asynch_io` auf `FALSE` gesetzt ist.

Der Wechsel einer Datenbankinstanz von gepufferten I/O-Operationen auf Direct-I/O kann auch zu einer schlechteren Performance führen, etwa wenn viele I/O-Operationen aus dem betriebssystemseitigen Buffercache bedient werden. Ein größerer Datenbank-Buffercache verhindert diese Probleme. Grundsätzlich muss die Datenbank gleich viel Memory zur Verfügung haben, wie das Betriebssystem vor der Umstellung für den Buffercache.

4.1.2.4 Zugriff

In einer Real-Application-Cluster-Konfiguration sind für die Speicherung von Datenbankfiles nur von Oracle zertifizierte Cluster-Filesysteme (CFS) möglich. Eine Übersicht, welche Plattformen welche CFS unterstützen, ist auf My Oracle Support im Artikel 183408.1, „Raw Devices and Cluster Filesystems With Real Application Clusters" dokumentiert. Es ist dabei zu beachten, dass gewisse CFS nur die Speicherung der Oracle Binaries unterstützen, nicht aber der Datenbankfiles.

4.1.3 Raw-Devices

Werden Raw-Devices eingesetzt, muss für jedes Datenbankfile ein entsprechendes Raw-Device auf Betriebssystemebene vorhanden sein. Aus diesem Grunde ist der Einsatz von Raw-Devices ohne Virtualisierungsschicht, welche die darunterliegenden physischen Disks virtualisiert, schwer denkbar. Abbildung 4.2 zeigt ein Beispiel über die Anwendung eines Logical Volume Managers. Jedes der vier physischen Volumes (PV) basiert auf zwei Disks, die hardwaremäßig gespiegelt sind (hier insgesamt 8 physische Disks). Jeder Tablespace besteht aus einem Datenfile, das auf einem Logical Volume (LV) basiert. Dieses

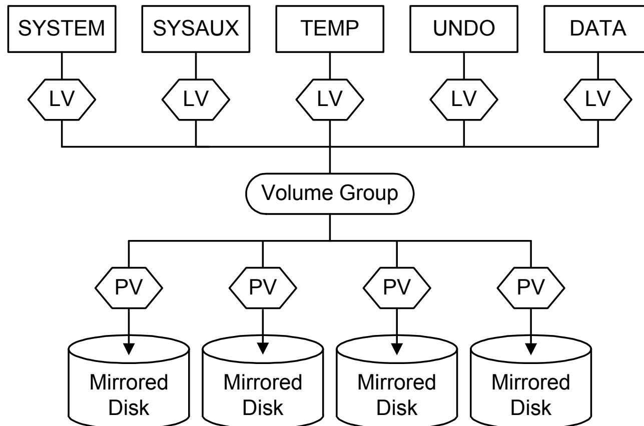

Abbildung 4.2
Beispiel einer Raw-Device-Konfiguration, die hardwaremäßige Spiegelung und Striping auf Software-Ebene nutzt

wiederum nutzt die Vorteile aller physischen Volumes, indem die Daten über diese gestriped werden.

Mit einer solchen Konfiguration werden die physischen Disks nicht direkt von der Datenbankinstanz angesprochen, was Änderungen auf der physischen Ebene ermöglicht, ohne das Setup auf Datenbankebene zu ändern.

In den folgenden Abschnitten beschreiben wir die Eigenschaften von Raw-Devices bezüglich Verwaltung, Verfügbarkeit, Performance und Zugriff.

4.1.3.1 Verwaltung

Operationen wie Hinzufügen, Entfernen oder Ersetzen einer Disk (inklusive Wiederverteilung der Daten) können nur dann unterbrechungsfrei (ohne Stoppen der Datenbankinstanz) erfolgen, wenn ein Logical Volume Manager, oder ein Netzwerkdevice, das solche Features unterstützt, eingesetzt werden.

Operationen wie Verschieben oder Kopieren von Datenbankfiles können – nach vorgängiger Benachrichtigung der Datenbankinstanz – mit Betriebssystembefehlen erfolgen. Im Gegensatz zu den Filesystemen sind die allgemein bekannten Befehle bei Raw-Devices nicht möglich. Für diese einfachen Operationen mit Raw-Devices ist spezielles Wissen notwendig. Im folgenden Beispiel, ausgeführt auf einem Linux-Server, zeigen wir, wie ein Datenbankfile von einem Device auf ein anderes verschoben wird. Für die Evaluation der Filegröße verwenden wir den dbfsize-Befehl (`dbfsize` ist Teil der Oracle Binaries) und für das Kopieren des Datenfiles kommt der dd-Befehl zum Einsatz (`dd` wird vom Betriebssystem zur Verfügung gestellt).

```
SQL> SELECT file_name
  2  FROM dba_data_files
  3  WHERE tablespace_name = 'TEST';

FILE_NAME
-----------------------
/dev/mapper/vg01-test1

SQL> ALTER TABLESPACE test OFFLINE;
```

```
SQL> !dbfsize /dev/mapper/vg01-test1

Database file: /dev/mapper/vg01-test1
Database file type: raw device
Database file size: 12800 8192 byte blocks

SQL> !dd if=/dev/mapper/vg01-test1 -
>        of=/dev/mapper/vg01-test2 -
>        count=12800 bs=8192
12800+0 records in
12800+0 records out

SQL> ALTER DATABASE
  2  RENAME FILE '/dev/mapper/vg01-test1'
  3  TO '/dev/mapper/vg01-test2';

SQL> ALTER TABLESPACE test ONLINE;
```

Bei der Planung einer solchen Operation ist zu berücksichtigen, dass bestimmte Betriebssysteme den ersten Teil eines Raw-Devices für die Verwaltung von Meta-Informationen reservieren. In diesem Fall ist ein Teil des Files beim Kopieren des Raw-Device-Inhalts zu überspringen. Diese Problematik lässt sich durch den Einsatz von RMAN (siehe Kapitel 13 „Backup und Recovery") einfach umgehen. Der folgende RMAN-Befehl führt die gleiche Operation aus, wie im vorhergehenden Beispiel.

```
RUN {
  ALLOCATE CHANNEL c TYPE disk;
  SQL 'ALTER TABLESPACE test OFFLINE';
  BACKUP AS COPY DATAFILE '/dev/mapper/vg01-test1'
                 FORMAT '/dev/mapper/vg01-test2';
  SWITCH DATAFILE '/dev/mapper/vg01-test1'
         TO DATAFILECOPY '/dev/mapper/vg01-test2';
  SQL 'ALTER TABLESPACE test ONLINE';
  RELEASE CHANNEL c;
}
```

Sind die Verfügbarkeitsanforderungen erhöht, bietet RMAN auch die Möglichkeit, die Kopie des Datenfiles online zu machen:

```
RUN {
  ALLOCATE CHANNEL c TYPE disk;
  BACKUP AS COPY DATAFILE '/dev/mapper/vg01-test1'
                 FORMAT '/dev/mapper/vg01-test2';
  SQL 'ALTER TABLESPACE test OFFLINE';
  SWITCH DATAFILE '/dev/mapper/vg01-test1'
         TO DATAFILECOPY '/dev/mapper/vg01-test2';
  RECOVER DATAFILE '/dev/mapper/vg01-test1';
  SQL 'ALTER TABLESPACE test ONLINE';
  RELEASE CHANNEL c;
}
```

Auch wenn das Vergrößern von Datenbankfiles auf Raw-Devices unterstützt wird, verzichtet man normalerweise darauf. Wer mehr Platz benötigt, weist üblicherweise dem neuen Datenfile ein neues Raw-Device zu.

4.1.3.2 Verfügbarkeit

Mit Ausnahme der Control-Files, der Redolog-Files und der archivierten Log-Files (welche auch softwaremäßig gespiegelt werden können) kann die Verfügbarkeit der Files auf

Raw-Devices nur durch den Einsatz von Spiegelung auf Hardware-Ebene, oder mit Hilfe eines Logical Volume Managers gewährleistet werden.

4.1.3.3 Performance

Raw-Devices benötigen keine spezielle Konfiguration für die optimale Performance. I/O-Operationen auf Datenfiles, die auf Raw-Devices liegen, können nicht vom betriebssystemseitigen Buffercache profitieren. Dies kann zu einer verminderten Performance führen, wenn Datenfiles von einem Filesystem auf ein Raw-Device verschoben werden, das vorher stark vom betriebssystemseitigen Buffercache profitiert hat. Um dies zu verhindern, sollte der Datenbank Buffercache entsprechend groß sein. Grundsätzlich muss die Datenbank gleich viel Memory besitzen, wie das Betriebssystem vor der Umstellung für den Buffercache verwendet hat.

4.1.3.4 Zugriff

In einer Real-Application-Cluster-Konfiguration stellt die Datenbankinstanz sicher, dass die Datenbankfiles auf konsistente und synchrone Art und Weise modifiziert sind. Daher kann eine Datenbank auf Raw-Devices gleichzeitig von mehreren Instanzen geöffnet werden. Dies ist auch der Grund dafür, wieso in der Vergangenheit fast alle geclusterten Datenbanken auf Raw-Devices verwaltet wurden.

4.1.4 Automatic Storage Management

Automatic Storage Management (ASM) ist ein Datenbank-Feature, das seit 10g eingeführt ist. ASM soll ohne zusätzliche Lizenzkosten die vier zentralen Anforderungen (Verwaltung, Verfügbarkeit, Performance und Zugriff) auf einfache, flexible und kostengünstige Art und Weise erfüllen. Anstelle einer vollständigen Beschreibung aller ASM-Features, konzentrieren wir uns in diesem Abschnitt auf die Möglichkeiten, um die erwähnten Anforderungen zu erfüllen. Die Installation und die Konfiguration von ASM ist in Kapitel 5 „Automatic Storage Management“ beschrieben.

4.1.4.1 Verwaltung

Operationen wie das Hinzufügen, Entfernen oder Ersetzen einer Disk (inklusive Wiederverteilung der Daten) kann online erfolgen, d.h. ohne die Datenbankinstanz zu stoppen, welche die modifizierte Diskgruppe verwendet. Die Wiederverteilung der Daten, das Rebalancing, erfolgt automatisch mit jeder Modifikation einer Diskgruppe. Es empfiehlt sich daher beim Ersetzen einer Disk die beiden Operationen (ADD und DROP) in einem einzigen ALTER DISKGROUP-Statement zusammenzufassen. Auf diese Weise erfolgt nur eine Rebalance-Operation. Das folgende Beispiel illustriert dieses Vorgehen:

```
ALTER DISKGROUP data
DROP DISK disk05
ADD FAILGROUP fg02 DISK '/dev/mapper/vg01-asmdisk09' NAME disk09
REBALANCE POWER 1
```

Mit der Klausel `REBALANCE POWER` kontrolliert man die Zahl der Hintergrundprozesse (ARB*n*), welche die Re-Distribution der Daten durchführen. Werte von 0 bis 11 sind erlaubt. Je höher der Wert, desto schneller ist das Rebalancing, desto höher sind aber auch die Auswirkungen auf das System. Der Default-Wert (1) wird über den Parameter `asm_power_limit` geändert. **Achtung:** der Wert 0 schaltet das automatische Rebalancing ganz aus.

Ab 11g lässt sich das Rebalancing beschleunigen, indem man die Diskgruppe im Restricted-Modus mountet. Weil in diesem Modus nur eine einzige Instanz die Diskgruppe mounten kann, fällt der Synchronisationsaufwand zwischen den Instanzen weg und das Rebalancing ist entsprechend schneller. Für ein solches Rebalancing wird normalerweise ein hoher Wert in der `REBALANCE POWER`-Klausel definiert.

Praxistipp

Ein Vorteil von ASM ist die Möglichkeit, Rebalancing-Operationen unabhängig von der physikalischen Lokation der Disks durchzuführen. So kann beispielsweise eine Datenbank unterbrechungsfrei von einem Storage-Subsystem auf ein anderes (z.B. auf ein neues SAN) verschoben werden. Der gleichzeitige Zugriff von ASM auf die Disks beider Speichersubsysteme ist die einzige Voraussetzung, die erfüllt sein muss.

Das Verschieben und Kopieren von Datenbankfiles wird normalerweise mit einem der folgenden Tools gemacht:

- RMAN (das Beispiel von Abschnitt 4.1.3.1 ist auch für ASM anwendbar)
- DBMS_FILE_TRANSFER-Package

Das Vergrößern von Datenbankfiles wird mit ASM transparent unterstützt. Einzige Voraussetzung ist genügend freier Platz in der entsprechenden Diskgruppe. In Abschnitt 4.2.2 und 4.2.3 zeigen wir je ein Beispiel für automatisches und manuelles Resizing.

4.1.4.2 Verfügbarkeit

ASM verfügt über drei verschiedene Diskgruppen-Typen: externe Redundanz, normale Redundanz und hohe Redundanz. Der Diskgruppen-Typ bestimmt den Level der Spiegelung. Mit externer Redundanz werden die Spiegelungseigenschaft des Speichersubsystem genutzt. Die anderen beiden Typen gewährleisten die Spiegelung von ASM. Normale Redundanz spiegelt die Daten doppelt, hohe Redundanz dreifach.

Wenn das Speichersubsystem ein High-End-SAN oder -NAS ist, kommen die Spiegelungs- (und Striping-)Möglichkeiten von ASM normalerweise nicht zur Anwendung, eine Ausnahme bilden Cluster, deren Knoten und Speichersubsysteme über mehrere Lokationen verteilt sind. In einem solchen Fall ist es vorteilhafter, die Spiegelung (und das Striping) dem Speichersubsystem zu überlassen. ASM nutzt dann die entsprechenden Disks in einer Diskgruppe mit externer Redundanz.

4.1.4.3 Performance

ASM benötigen keine spezielle Konfiguration für die optimale Performance – einzige Ausnahme ist, wenn die Disks und die Instanzen über zwei (oder drei) Lokationen verteilt sind. In diesem Fall sollte man nicht nur für jede Lokation eine „Failure Group“ (welche die lokalen Disks beinhaltet) definieren, sondern auch die Instanzen so einrichten, dass die „Preferred Failure Group“ lokal ist. Für diesen Zweck steht der Parameter `asm_preferred_read_failure_groups` zur Verfügung.

4.1.4.4 Zugriff

ASM wurde sowohl für den Einsatz in Real-Application-Clusters- als auch für Single-Instanz-Umgebungen entwickelt. Die einzige Voraussetzung beim Einsatz in RAC-Umgebungen ist, dass der Zugriff auf die Disks für alle Server im Clusterverbund gewährleistet ist. Weil sich die Disks dem ASM als Raw-Devices präsentieren, sollte dies grundsätzlich kein Problem sein.

4.1.5 Die Auswahl der Datenbank-Speicheroption

Früher waren Raw-Devices bezüglich Performance und Eignung für Clustersysteme die erste Wahl waren. Mit ASM steht heute eine viel einfachere und flexiblere Lösung zur Verfügung. Hinzu kommt, dass Oracle plant, Raw-Devices in Zukunft nicht mehr zu unterstützen (siehe My Oracle Support Artikel 754305.1 „Announcement on using Raw devices with release 11.2“). Aus diesem Grunde empfehlen wir, in neuen Systemen Raw-Devices nicht mehr einzusetzen.

Um die vier zentralen Anforderungen (Verwaltung, Verfügbarkeit, Performance und Zugriff) zu erfüllen, ohne einen Logical-Volume-Manager eines Drittherstellers lizensieren zu müssen, empfehlen wir ASM. Wer Real Application Clusters mit der Standard Edition verwendet, muss zwingend ASM einsetzen.

Auch wenn die Verwaltung von ASM nicht sehr schwierig ist, ist ASM eine zusätzliche Infrastrukturkomponente, die verwaltet und verstanden werden muss. Wenn der zusätzliche Aufwand für Verwaltung und Setup den Nutzen gegenüber einem Filesystem übersteigt, empfehlen wir, anstelle von ASM ein Filesystem zu verwenden.

4.2 Data-, Temp- und Redolog-File-Attribute

Im vorherigen Abschnitt haben wir gezeigt, wie die Datenbankfiles im Storage-System gespeichert sind. In diesem Abschnitt beschreiben wir die Attribute, die für Datafiles, Tempfiles und Redolog-Files spezifiziert werden können.

Das einzige Attribut, das man für alle drei Filetypen definieren kann, ist die initiale Filegröße (Initial Size). Zusätzlich lassen sich Datafiles und Tempfiles (vorausgesetzt, man verwendet keine Raw-Devices) so spezifizieren, dass sie sich bei Bedarf automatisch vergrößern. Natürlich kann man die Datafiles auch manuell vergrößern.

4.2.1 Initial Size

Beim Erstellen eines Datafiles, eines Tempfiles oder eines Redolog-Files wird die initiale Größe mit der `SIZE`-Klausel definiert. Wie das folgende Beispiel zeigt, kann man die Größe mit dem entsprechenden Suffix „K“, „M“, „G“, „T“, „P“ oder „E“ sowohl in Bytes, Kilobytes, Megabytes, Gigabytes, Terabytes, Petabytes oder Exabytes spezifizieren.

```
SQL> CREATE TEMPORARY TABLESPACE test
  2  TEMPFILE '/u00/oradata/ora11g/test01ora11g.dbf' SIZE 1G;
```

Bei der Erstellung eines Datenfiles oder eines Redolog-Files initialisiert die Database-Engine alle zugehörigen Blöcke. Aus diesem Grund kann die Erstellung eines großen Tablespaces auch entsprechend viel Zeit beanspruchen. Eine vorgängige Initialisierung findet für Tempfiles nicht zwingend statt, sondern es entsteht, abhängig vom Betriebssystem, ein sogenanntes „Sparsefile“. Der erforderliche Platz für einen Block in einem Sparsefile wird erst dann alloziert, wenn man den Block das erste Mal verwendet. Dies bedeutet, dass der erforderliche Diskplatz für ein Tempfile erst bei dessen Nutzung vollständig alloziert wird. Das folgende Beispiel illustriert dieses Verhalten, wir verwenden dazu den zuvor erstellten Tablespace:

1. Anzeige der Dateigröße des Tempfiles mittels Betriebssystem-Kommando (in diesem Fall Linux). Wie aus dem folgenden Output ersichtlich ist, zeigt der Befehl `ls -l` eine Filegröße von 1 GB an (eigentlich 1 GB + ein Datenbankblock). Gleichzeitig zeigt uns der Befehl `ls -s` aber lediglich eine Filegröße von 129 Blöcken à 8192 Bytes (= 1‘056‘768 Bytes) an. Daraus folgt: Der erste Befehl zeigt die Dateigröße und der zweite Befehl den allozierten Platz an.

```
$ cd /u00/oradata/ora11g/
$ ls -l test01ora11g.dbf
-rw-r----- 1 oracle oinstall 1073750016 Aug 5 18:17 test01ora11g.dbf
$ ls -s --block-size=8192 test01ora11g.dbf
129 test01ora11g.dbf
```

2. Nun erstellen wir im temporären Tablespace ein temporäres Segment mit ca. 100 MB Daten. Danach löschen wir das Objekt wieder.

```
SQL> CREATE GLOBAL TEMPORARY TABLE t (id NUMBER, pad VARCHAR2(1000))
  2  TABLESPACE test;

SQL> INSERT INTO t
  2  SELECT rownum, rpad('*',1000,'*')
  3  FROM dual
  4  CONNECT BY level <= 100000;

SQL> DROP TABLE t PURGE;
```

3. Jetzt werten wir den Platz nochmals wie in Punkt 1 aus. Beachten Sie, dass nun 11‘335 Blöcke alloziert sind – das Tempfile verwendet nun ca. 93 MB Platz auf Betriebssystemebene.

```
$ ls -s --block-size=8192 test01ora11g.dbf
11335 test01ora11g.dbf
```

Im Zusammenhang mit Sparsefiles gibt es zwei Probleme. Erstens: Weil der Platz erst bei der effektiven Nutzung alloziert wird, kann es vorkommen, dass der Platz auf Betriebssystemebene vorgängig bereits anderweitig verwendet wurde und für das Sparsefile nicht mehr zur Verfügung steht. Dies führt ggf. zu einem Fehler (z.B. ORA-01114: IO error writing block to file). Zweitens kann es zu Performance-Einbußen führen, weil auf bestimmten Betriebssystemen für Sparsefiles nicht alle Features unterstützt werden (beispielsweise unterstützt Solaris kein Direct-I/O).

Praxistipp

Wir empfehlen, keine Sparsefiles zu verwenden und die Tempfiles vorgängig, das heißt bevor die Zuordnung zur Datenbank erfolgt, auf Betriebssystem-Ebene zu erstellen.

Das folgende Beispiel zeigt, wie man Sparefiles beim Erstellen von Tempfiles verhindern kann.

1. Erstellen des Files auf Betriebssystem-Ebene (in diesem Fall Linux). Dazu kann das Unixprogramm `dd` verwendet werden. Beachten Sie, dass die Größe der Datei mit der Größe des Tempfiles übereinstimmen muss.

```
$ cd /u00/oradata/ora11g
$ dd if=/dev/zero of=test01ora11g.dbf bs=1024 count=1048584
```

2. Zuweisen des erstellten Files zur Datenbank. Dabei sind zwei Dinge zu beachten: Erstens die `REUSE`-Klausel zu verwenden, weil die Datei ja bereits existiert. Zweitens die `SIZE`-Klausel nicht zu spezifizieren, da die Datenbank die Dateigröße vom Betriebssystem übernimmt.

```
SQL> CREATE TEMPORARY TABLESPACE test
  2  TEMPFILE '/u00/oradata/ora11g/test01ora11g.dbf' REUSE;
```

4.2.2 Automatische Filevergrößerung

Datafiles und Tempfiles lassen sich mit oder ohne automatischer Filevergrößerung (Autoextend) erstellen. Defaultmäßig ist die automatische Filevergrößerung nicht aktiviert – vorausgesetzt die `SIZE`-Klausel ist spezifiziert, oder eine bestehende Datei wird wieder verwendet. Dies bedeutet, dass der DBA die Verantwortung für die manuelle Vergrößerung der Datafiles trägt, falls zusätzlicher Platz erforderlich ist. Das manuelle Vergrößern ist im nächsten Abschnitt beschrieben. Die automatische Filevergrößerung vermeidet das manuelle Eingreifen. Bei deren Aktivierung wird definiert, in welchen Schritten die Vergrößerung erfolgen soll und wie groß die Datei maximal sein darf. Das folgende Beispiel zeigt die Spezifikation eines Tempfiles mit automatischer Dateivergrößerung:

```
SQL> CREATE TEMPORARY TABLESPACE test
  2  TEMPFILE '/u00/oradata/ora11g/test01ora11g.dbf' SIZE 1G
  3  AUTOEXTEND ON NEXT 100M MAXSIZE UNLIMITED;
```

Die automatische Filevergrößerung ist nur interessant, wenn die Platzüberwachung anstelle auf Tablespace-Ebene auf Filesystem- oder auf Diskgroup-Ebene erfolgen soll.

Praxistipp

Für Tempfiles empfehlen wir, die automatische Filevergrößerung auszuschalten. Fehlerhafte SQL-Anweisungen, die eine große Menge temporärer Informationen generieren, können einen temporären Tablespace sehr schnell füllen. Bei temporären Tablespaces ist es deshalb sinnvoller, eine Fehlermeldung an die verursachende Session zu senden, als unnötigerweise eines oder mehrere Tempfiles zu erweitern.

4.2.3 Manuelle Filevergrößerung

Data- und Tempfiles können bei Bedarf jederzeit vergrößert werden – vorausgesetzt der dazu erforderliche Platz ist im Speichersystem verfügbar. Im folgenden Beispiel zeigen wir den entsprechenden Befehl `RESIZE` :

```
SQL> ALTER DATABASE
  2  TEMPFILE '/u00/oradata/ora11g/test01ora11g.dbf' RESIZE 2G;
```

Der Befehl `RESIZE` kann Data- und Tempfiles auch verkleinern. Das kann jedoch nur dann erfolgen, wenn der zu verkleinernde Teil des Files keine Extents enthält. Das folgende Beispiel zeigt, was geschieht, wenn diese Voraussetzung nicht erfüllt ist:

```
SQL> ALTER DATABASE
  2  TEMPFILE '/u00/oradata/ora11g/test01ora11g.dbf' RESIZE 1G;
ALTER DATABASE
*
ERROR at line 1:
ORA-03297: file contains used data beyond requested RESIZE value
```

Beachten Sie, dass der freie Platz im File nicht entscheidend ist. Es kommt auf die Position der Extents innerhalb der Datei an. Auch wenn nur ein einzelnes, kleines Extent am Ende der Datei vorhanden ist, kann die Resize-Operation nicht ausgeführt werden.

Praxistipp

Das Skript `lstsres.sql` kann für die Anzeige der Platzverhältnisse (Größe, benutzter Platz, Verkleinerungspotenzial ohne Reorganisation) in den Datenfiles eines bestimmten Tablespaces verwendet werden. Ist der freie Platz im Datenfile wesentlich größer, als der Platz, der mit einem Resize gewonnen werden kann, sind Segmente gelöscht worden. Werden keine neuen Segmente in diesem Tablespace angelegt, ist es sinnvoll die Segmente zu verschieben, sodass mittels Resize das Datenfile verkleinert werden kann (siehe Abschnitt 4.6.2).

4.3 Extent Management Optionen

Bei der Erstellung eines Tablespaces wird definiert, wie die Extents darin verwaltet werden sollen. Das heißt, es ist festgelegt, wie der freie Platz innerhalb des Tablespaces verwaltet wird. Es gibt grundsätzlich zwei Verwaltungsarten: Dictionary Managed Tablespaces und Locally Managed Tablespaces.

Ein Extent ist eine Abfolge von Blöcken innerhalb einer Datei. Weil ein Extent sich nicht über mehrere Dateien erstrecken darf und auch keine “Löcher” aufweisen kann, ist es durch

seinen Startblock und seine Größe definiert. Zwei Anforderungen müssen von der Datenbank bezüglich Extentverwaltung erfüllt sein. Erstens: Ein bestimmter Block (und demzufolge ein Teil eines Datafiles) kann nur einem einzigen Extent zugeordnet sein. Zweitens: Wird ein Extent nicht mehr benutzt, ist der freiwerdende Teil des Datafiles für andere Extents wieder zur Verfügung zu stellen. Während die erste Anforderung einfach zu implementieren ist, verhält es sich mit der zweiten etwas komplizierter. Sind die Extents unterschiedlich groß (nicht uniform), kann der freigewordene Platz nicht in allen Fällen wiederverwendet werden. Oder mit anderen Worten: Ungleiche Extentgrößen können zu Fragmentierung führen.

Im folgenden Abschnitt beschreiben wir, wie die beiden oben erwähnten Anforderungen an das Extent Management mit Dictionary Managed und Locally Managed Tablespaces implementiert sind. Zuerst zeigen wir aber, was eine Extent-Map ist, und werden einige wichtige Konzepte betreffend Dimensionierung und Allokation von Extents betrachten. Zum Schluss geben wir ein paar praktische Hinweise betreffend Auswahl und Einsatz der verfügbaren Verwaltungsoptionen.

4.3.1 Extent Map

Zu Beginn dieses Kapitels haben wir gesehen, dass ein Segment aus einem oder mehreren Extents besteht. Die Information über Anzahl, Größe und Speicherort dieser Extents wird in einer sogenannten „Extent Map“ verwaltet (siehe Abbildung 4.3). Die Extent-Map ist im Segment selber gespeichert. Besteht das Segment aus einigen Hundert Extents, kann die Extent-Map vollständig im Segmentheader (siehe Kapitel 2 „Architektur und Administration“) gespeichert werden; etwa die Hälfte der Blöcke ist für diesen Zweck reserviert. Ist nicht genügend Platz im Segmentheader vorhanden, wird die Extent-Map über die notwendige Anzahl Blöcke verteilt.

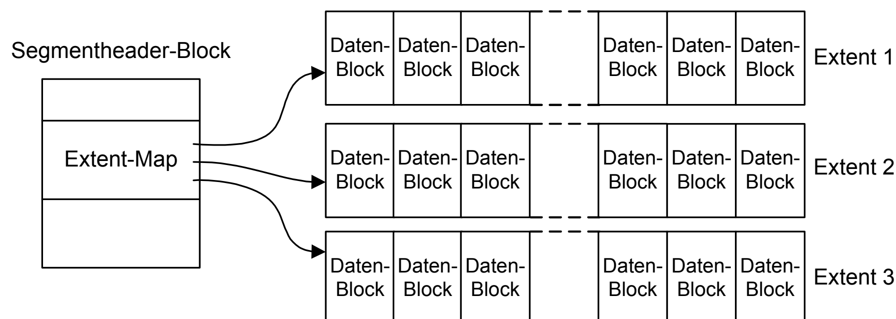

Abbildung 4.3 Die Extent-Map beinhaltet Ort und Größe aller Extents, die zum entsprechenden Segment gehören

Neben anderen Aufgaben verwendet die Database-Engine die Extent-Map für die Ausführung von Full-Scans. Beim Full-Scan werden die Extents in der gleichen Reihenfolge gelesen, wie sie in der Extent-Map gespeichert sind.

4.3.2 Storage-Parameter

Die Storage-Klausel definiert über diverse Attribute die Größe und Anzahl der Extents. Weil die Bedeutung und auch die Validierung dieser Attribute von der Extentverwaltungsmethode abhängig sind, betrachten wir in diesem Abschnitt nur die allgemeine Bedeutung dieser Attribute. Die spezifische Implementierung wird zusammen mit den Extentverwaltungsoptionen später in diesen Kapitel beschrieben.

Die folgenden drei Storageparameter bestimmen die Extentgröße:

- `INITIAL`: Die Größe des ersten Extents eines Segments
- `NEXT`: Die Größe der Extents, die nach dem ersten Extent erstellt werden
- `PCTINCREASE`: Der Prozentsatz, zu dem das dritte und alle folgenden Extents gegenüber dem vorgängigen Extent wachsen. Ein Beispiel: Die Größe des dritten Extents ist gleich `NEXT`*(1+`PCTINCREASE`/100).

Zwei weitere Storage-Parameter bestimmen die Anzahl Extents:

- `MINEXTENTS`: Die Anzahl Extents, die bei der Erstellung des Segments alloziert werden
- `MAXEXTENTS`: Die maximale Anzahl Extents, die ein Segment haben kann. Das Schlüsselwort `UNLIMITED` definiert keine obere Grenze.

Das folgende SQL-Statement zeigt, wie man diese Parameter für eine Tabelle definiert:

```
SQL> CREATE TABLE t (id NUMBER, pad VARCHAR2(1000))
  2  STORAGE (
  3    INITIAL 10M
  4    NEXT 1M
  5    PCTINCREASE 0
  6    MINEXTENTS 1
  7    MAXEXTENTS UNLIMITED
  8  );
```

Wie wir später in diesem Kapitel sehen werden, kann man für Dictionary Managed Tablespaces eine Defaultstorage-Klausel spezifizieren. Diese Klausel bestimmt die Defaulteinstellung für alle Segmente, für die keine explizite Storageklausel besteht.

4.3.3 Extent-Allozierung

In den meisten Situationen ist die Allokation von Extents eine triviale Operation. Benötigt ein Segment mehr Platz als bereits alloziert, muss ein neues Extent alloziert und diesem Segment zugeordnet werden. Es gibt jedoch zwei spezielle Situationen, die wir an dieser Stelle diskutieren wollen: die verzögerte (deferred) Segmenterstellung sowie parallele Inserts.

Bis 11g R1 wurden gleichzeitig mit dem Erstellen eines Objekts immer auch die initialen Extents des Segmentes alloziert. Mit 11g R2 hat sich dieses Verhalten dank des Features „Deferred Segment Creation“ geändert. Ein Segment und seine initialen Extents werden erst dann alloziert, wenn der erste Eintrag (Einfügen eines Wertes) in die Tabelle erfolgt. Das folgende Beispiel illustriert dieses Verhalten:

```
SQL> CREATE TABLE t (n NUMBER);

Table created.

SQL> SELECT segment_created
  2  FROM user_tables
  3  WHERE table_name = 'T';

SEGMENT_CREATED
---------------
NO

SQL> SELECT segment_name
  2  FROM user_segments
  3  WHERE segment_name = 'T';

no rows selected

SQL> INSERT INTO t VALUES (1);

1 row created.

SQL> SELECT segment_created
  2  FROM user_tables
  3  WHERE table_name = 'T';

SEGMENT_CREATED
---------------
YES

SQL> SELECT segment_name
  2  FROM user_segments
  3  WHERE segment_name = 'T';

SEGMENT_NAME
------------
T
```

Dieses Verhalten lässt sich mit dem Datenbankparameter `DEFERRED_SEGMENT_CREATION` überschreiben. Dazu muss der (Default-)Wert `TRUE` auf `FALSE` gesetzt sein. Dies kann auf System- oder auf Sessionebene erfolgen, oder beim Erstellen der Tabelle mit der Klausel `SEGMENT CREATION IMMEDIATE`.

Folgende Einschränkungen sind bezüglich „Deferred Segment Creation" zu beachten:

- Version 11.2.0.1 unterstützt nur nichtpartitionierte Heap-Tabellen und deren abhängige Objekte (z.B. Indizes und LOBs). Diese Einschränkung ist mit Version 11.2.0.2 aufgehoben.
- Segmente, die zu SYS, SYSTEM, PUBLIC, OUTLN, oder XDB gehören, werden nicht unterstützt.
- Bitmap-Join-Indizes und Domain-Indizes sind nicht unterstützt.
- Segmente, die in Dictionary Managed Tablespaces verwaltet werden, sind ebenfalls nicht unterstützt.

Beim Arbeiten mit „Deferred Segment Creation" muss der Tablespace ausreichend dimensioniert sein, um alle neuen Segmente zu speichern, beziehungsweise wenn auf Dateiebene Autoextend zum Einsatz kommt, genügend Platz auf Betriebssystemebene vorhanden ist).

Ist dies nicht sichergestellt, können Fehler auftreten, sobald die Datenbank versucht, ein neues Extent zu allozieren.

Eine Eigenheit von parallelen Inserts ist, dass der verfügbare freie Platz in den bereits allozierten Extents bei der Ausführung nicht berücksichtigt wird. Dies bedeutet, dass für jeden parallelen Slaveprozess, welcher die Insertoperation durchführt, mindestens ein neues Extent alloziert und dem geänderten Tabellensegment zugeordnet wird. Das folgende Beispiel illustriert dieses Verhalten. Beachten Sie, wie die zweite Insertanweisung, die zehn Einträge parallel in die Tabelle einfügt, zur Allozierung von zwei neuen Extents führt, und dies obwohl noch genügend Platz im ersten Extent vorhanden ist.

```
SQL> CREATE TABLE t (n NUMBER);

Table created.

SQL> INSERT INTO t VALUES (1);

1 row created.

SQL> COMMIT;

Commit complete.

SQL> SELECT extent_id
  2  FROM user_extents
  3  WHERE segment_name = 'T';

 EXTENT_ID
----------
         0

SQL> ALTER SESSION ENABLE PARALLEL DML;

Session altered.

SQL> INSERT /*+ parallel(t,2) */ INTO t
  2  SELECT rownum+1 FROM dual CONNECT BY level <= 10;

10 rows created.

SQL> COMMIT;

Commit complete.

SQL> SELECT extent_id
  2  FROM user_extents
  3  WHERE segment_name = 'T';

 EXTENT_ID
----------
         0
         1
         2
```

4.3.4 Dictionary Managed Tablespaces

Die Extent-Informationen von Dictionary Managed Tablespaces sind in zwei Data-Dictionary-Tabellen verwaltet. Die erste Tabelle (`uet$`) beinhaltet für jedes benutzte Extent einen Eintrag. Die zweite Tabelle (`fet$`) beinhaltet einen Eintrag für jedes unbenutzte Extent. Wird ein neues Extent alloziert, muss Oracle freien Platz in `fet$` finden, darin eine Row aktualisieren oder löschen und danach eine Row in `uet$` einfügen. Um Platz freizugeben, der einem Extent zugeordnet ist, wird eine Row aus `uet$` gelöscht und in `fet$` eingefügt. Aufgrund dieser Methode können Operationen, die viele Extents ändern, wie beispielsweise das Löschen einer Tabelle mit Tausenden von Extents, sehr zeitintensiv sein.

Dictionary Managed Tablespaces unterstützen alle Storageparameter. Sie funktionieren wie in Abschnitt 4.3.2 beschrieben. Dies führt oft dazu, dass Extents, die in einem Dictionary Managed Tablespace erstellt wurden, keine einheitliche Größe aufweisen, was zu Fragmentierung führt.

Praxistipp

In einem Dictionary Managed Tablespaces kann man Fragmentierung verhindern, indem die Extentgröße ein Vielfaches eines bestimmten Werts beträgt. Für diesen Zweck steht die Klausel `MINIMUM EXTENT` zur Verfügung.

Das folgende Beispiel zeigt, wie man einen Dictionary Managed Tablespace erstellt, dessen Extents ein Vielfaches von 1 MB betragen.

```
SQL> CREATE TABLESPACE test
  2  DATAFILE '/u00/oradata/ora11g/test01ora11g.dbf' SIZE 1G
  3  EXTENT MANAGEMENT DICTIONARY
  4  MINIMUM EXTENT 1M;
```

Beim Erstellen eines Dictionary Managed Tablespace sind Default-Storage-Parameter für Segmente möglich, denen keine expliziten Storage-Parameter zugewiesen sind. Hierzu ein Beispiel:

```
SQL> CREATE TABLESPACE test
  2  DATAFILE '/u00/oradata/ora11g/test01ora11g.dbf' SIZE 1G
  3  EXTENT MANAGEMENT DICTIONARY
  4  DEFAULT STORAGE (
  5    INITIAL 4M
  6    NEXT 2M
  7    PCTINCREASE 0
  8    MINEXTENTS 1
  9    MAXEXTENTS UNLIMITED
 10  );
```

Folgende Einschränkungen beziehen sich auf Dictionary Managed Tablespaces:

- Deferred Segment Creation ist nicht unterstützt.
- Automatic Freespace Management ist nicht unterstützt. Abschnitt 4.4.3 behandelt dieses Feature.

4.3.5 Locally Managed Tablespaces

Die Extent-Informationen von Locally Managed Tablespaces sind in sogenannten „Space-Blöcken“ verwaltet, die innerhalb des Tablespaces gespeichert werden. Mit anderen Worten: Die Extentverwaltung findet nicht mehr im Data Dictionary statt, sondern lokal im Tablespace. Um die Größe der Extents unter Kontrolle zu halten, kann man zwischen Uniform-Size oder Auto-Allocate wählen. Im zweiten Fall überlässt man die Wahl der passenden Extentgröße dem System. Zudem ist die Wahl zwischen Smallfile- und Bigfile-Tablespaces möglich.

4.3.5.1 Uniform Extent Size

Wie der Name sagt, haben mit der Uniform Extent Size alle Extents exakt dieselbe Größe. Dies hat den Vorteil, dass keine Fragmentierung entstehen kann. Auf der anderen Seite wird der allozierte Platz unter Umständen – speziell bei großen Extents – nicht optimal genutzt. De facto ist für jedes Segment mindestens ein Extent zu allozieren, unabhängig davon, wie viele Daten darin gespeichert werden müssen.

Praxistipp

Um eine Überallozierung zu verhindern, sollte die Extentgröße ein Vielfaches kleiner sein, als die für das Segment erwartete Datenmenge. Zudem sollte bei parallelen Inserts die Extentgröße ein Vielfaches kleiner sein, als die durch eine Ausführung eines einzelnen Slaveprozesses eingeführte Datenmenge.

Das folgende Beispiel zeigt die Erstellung eines Tablespace mit einer Uniform Size von 1 MB:

```
SQL> CREATE TABLESPACE test
  2  DATAFILE '/u00/oradata/ora11g/test01ora11g.dbf' SIZE 1G
  3  EXTENT MANAGEMENT LOCAL UNIFORM SIZE 1M;
```

Die Verwendung der Uniform Extent Size berücksichtigt die in Abschnitt 4.3.2 erwähnten Speicherparameter nur bei der Segmenterstellung. Es gibt auch keine Einschränkung bezüglich der Anzahl Extents (`MAXEXTENTS` wird beispielsweise ignoriert). Das folgende Beispiel zeigt die Auswirkung der Uniform Extent Size bei der Erstellung eines Segments. Beachten Sie, dass gemäß `MINEXTENTS` drei Extents alloziert sein sollten. Gemäß dem `INITIAL`-Parameter sollte das erste Extent 4 MB groß sein, das zweite Extent gemäß dem `NEXT`-Parameter 2 MB betragen und das dritte Extent gemäß `NEXT` und `PCTINCREASE` 3 MB groß sein. Mit einer Uniform-Size von 1 MB sollte dies in Summe eine allozierte Größe von 9 MB ergeben. Beachten Sie auch, dass im Data Dictionary angepasste Werte vorhanden sind und nicht die in der SQL-Anweisung spezifizierten Werte.

```
SQL> CREATE TABLE t (id NUMBER, pad VARCHAR2(1000))
  2  STORAGE (
  3    INITIAL 4M
  4    NEXT 2M
  5    PCTINCREASE 50
  6    MINEXTENTS 3
  7    MAXEXTENTS 6
```

```
  8  )
  9  TABLESPACE test;

SQL> SELECT bytes, count(*)
  2  FROM user_extents
  3  WHERE segment_name = 'T'
  4  GROUP BY bytes;

     BYTES   COUNT(*)
---------- ----------
   1048576          9

SQL> SELECT initial_extent, next_extent, pct_increase,
  2         min_extents, max_extents
  3  FROM user_tables
  4  WHERE table_name = 'T';

INITIAL_EXTENT NEXT_EXTENT PCT_INCREASE MIN_EXTENTS MAX_EXTENTS
-------------- ----------- ------------ ----------- -----------
       9437184     2097152            0           1  2147483645
```

Das Beispiel zeigt, wie die Storage-Parameter die Extent-Allozierung beeinflussen. Wir empfehlen nicht, die Parameter in der Praxis auf diese Art und Weise zu spezifizieren.

Praxistipp

Beim Einsatz von Locally Managed Tablespaces mit Uniform Extent Size sollten die Storageparameter der Segmente nicht spezifiziert sein.

Für Locally Managed Tablespaces mit Uniform Extent Size gelten folgende Einschränkungen:

- Diese Extentverwaltungsoption wird für Undo-Tablespaces nicht unterstützt.
- Die Default-Storage-Klausel ist nicht unterstützt.

4.3.5.2 System Managed Extent Size

Das Ziel der System Managed Extent Size ist die Abstimmung der Extentgrößen auf die Größe des Segments, dem die Extents zugeordnet sind. Mit anderen Worten: Ein großes Segment sollte große Extents erhalten, ein kleines Segment kleine. Auf diese Weise sollte es möglich sein, sowohl Fragmentierung, als auch eine sehr große Anzahl Extents weitestgehend zu verhindern. Zu diesem Zweck verwendet die Database-Engine Standard-Extentgrößen von 64 KB, 1 MB, 8 MB oder 64 MB. Das folgende Beispiel zeigt, wie man einen Tablespace mit System Managed Extent Size erstellt:

```
SQL> CREATE TABLESPACE test
  2  DATAFILE '/u00/oradata/ora11g/test01ora11g.dbf' SIZE 1G
  3  EXTENT MANAGEMENT LOCAL AUTOALLOCATE;
```

Mit der System Managed Extent Size werden die Storage-Parameter auf die gleiche Art und Weise behandelt wie mit der Uniform Extent Size (Details siehe Abschnitt 4.3.5.1 „Uniform Extent Size“). Mit anderen Worten: Die Storage-Parameter sind mit einer Ausnahme lediglich für die initiale Dimensionierung des Segments notwendig.

Ab Version 11.1.0.7 wird das NEXT-Attribut bei parallelen Inserts für die Dimensionierung der Extents verwendet (zur Erinnerung: parallele Inserts allozieren immer neue Extents). In diesem Zusammenhang ist jedoch (ein weiterer) Spezialfall zu beobachten: die Allozierung von Extents, die nicht der Standart-Extentgröße entsprechen. Zur Verhinderung von nicht wiederverwendbarem freiem Platz, lassen sich die Extents auf einen Bruchteil der Standardgröße (64 KB, 1 MB, 8 MB oder 64 MB) verkleinern, wie das folgende Beispiel zeigt:

```
SQL> CREATE TABLE t (n NUMBER)
  2  STORAGE (INITIAL 1M NEXT 1M)
  3  TABLESPACE test;

Table created.

SQL> INSERT INTO t VALUES (1);

1 row created.

SQL> COMMIT;

Commit complete.

SQL> SELECT extent_id, bytes
  2  FROM user_extents
  3  WHERE segment_name = 'T';

 EXTENT_ID      BYTES
---------- ----------
         0    1048576

SQL> ALTER SESSION ENABLE PARALLEL DML;

Session altered.

SQL> INSERT /*+ parallel(t,2) */ INTO t
  2  SELECT rownum+1 FROM dual CONNECT BY level <= 200000;

200000 rows created.

SQL> COMMIT;

Commit complete.

SQL> SELECT extent_id, bytes
  2  FROM user_extents
  3  WHERE segment_name = 'T';

 EXTENT_ID      BYTES
---------- ----------
         0     589824
         1    1048576
         2     327680
         3    1048576
         4     327680
```

Für Locally Managed Tablespaces mit System Managed Extent Size gelten folgende Einschränkungen:

- Diese Art der Extentverwaltung ist für temporäre Tablespaces nicht unterstützt.
- Die Default Storage-Klausel ist nicht unterstützt.

4.3.5.3 Smallfile- vs. Bigfile-Tablespaces

Traditionsgemäß verwendet Oracle Smallfile-Tablespaces. Damit kann ein Tablespace typischerweise aus bis zu 1022 Files à 2^{22}-1 Blöcken bestehen. Die Limitierung ist betriebssystemabhängig; die erwähnten Werte sind für die am häufigsten eingesetzten Betriebssysteme gültig. Mit anderen Worten: Die maximale Größe eines Tablespaces mit einer Default-Blockgröße von 8 KB beträgt ca. 32 TB. Eine weitere Einschränkung ist die Anzahl der Datenbankfiles pro Datenbank. Diese wird mit dem Initialisierungsparameter `db_files` beschränkt. Maximal sind 65.533 Files pro Datenbank möglich. Insbesondere die zweite Einschränkung stellt für die meisten Datenbanken kein Problem dar, außer Sie planen eine sehr große Datenbank: Mit einer Blockgröße von 8 KB sind lediglich 64 Tablespaces à 32 TB möglich. Diese Limitierung fällt mit dem Einsatz von Bigfile-Tablespaces weg. Damit kann der Tablespace selber nicht größer werden, aber weil Bigfile-Tablespaces nur aus einem File bestehen, sind bis zu 65.533 Tablespaces à 32 TB vorstellbar.

Defaultmäßig erzeugt die Database-Engine Smallfile-Tablespaces. Der Default kann auf Datenbankebene entweder über die `SET DEFAULT TABLESPACE`-Klausel in der `CREATE DATABASE`-Anweisung geändert werden, oder für eine bereits bestehende Datenbank mit folgendem SQL-Befehl:

```
SQL> ALTER DATABASE SET DEFAULT BIGFILE TABLESPACE;
```

Die View `database_properties` zeigt die Default-Einstellung:

```
SQL> SELECT property_value
  2  FROM database_properties
  3  WHERE property_name = 'DEFAULT_TBS_TYPE';

PROPERTY_VALUE
---------------
BIGFILE
```

Beim Erstellen eines neuen Tablespaces mit dem Statement `CREATE TABLESPACE`, kann man den Datenbank-Default mit der Klausel `SMALLFILE` oder `BIGFILE` überschreiben. Die folgende SQL-Anweisung zeigt diese Möglichkeit:

```
SQL> CREATE SMALLFILE TABLESPACE test
  2  DATAFILE '/u00/oradata/ora11g/test01ora11g.dbf' SIZE 1G;
```

Für Bigfile-Tablespaces gilt folgende Einschränkung:

- Bigfile Dictionary Managed Tablespaces sind nicht unterstützt. Mit anderen Worten: Bigfile Tablespaces müssen “locally-managed” sein.

4.3.6 Auswahl der Extent-Management-Optionen

Locally Managed Tablespaces haben weitaus mehr Vorteile gegenüber Dictionary Managed Tablespaces und sind deshalb die erste Wahl beim Erstellen eines neuen Tablespaces. Auch Oracle empfiehlt nur noch die Erstellung von Locally Managed Tablespaces; es ist geplant das Erstellen von neuen Dictionary Managed Tablespaces in Zukunft nicht mehr zu unterstützen.

Praxistipp

In einer neuen Datenbank sollte der SYSTEM-Tablespace nur noch dann „Dictionary Managed“ sein, wenn dictionary-managed Transportable Tablespaces angehängt und im Read/Write-Modus betrieben werden. Wenn der SYSTEM-Tablespace „Locally Managed“ ist, müssen alle anderen Tablespaces im Read/Write-Modus locally-managed sein.

Die Wahl zwischen Uniform Extent Size und System Managed Extent Size hängt davon ab, ob die Größe des im Tablepace zu speichernden Segmentes bekannt ist oder nicht, und ob man in der Lage ist, geeignete `INITIAL`- und `NEXT`-Werte für jedes Segment zu spezifizieren.

Praxistipp

Ist die Segmentgröße bekannt und liegen keine speziellen Anforderungen für das `INITIAL`- und `NEXT`-Extent vor, sollte man ein Tablespace mit Uniform Extent Size verwenden. Werden Segmente mit sehr unterschiedlicher Größe erstellt, sind Tablespaces mit entsprechend unterschiedlicher Uniform Extent Size (beispielsweise ein Tablespace mit 64 KB und ein anderer mit 8 MB) möglich. Damit können die Segmente entsprechend ihrer Größe den Tablespaces zugeordnet sein.

Für Segmente, deren Größe nicht bekannt ist, oder deren `INITIAL`- und `NEXT`-Attribute unkritisch sind, sollte man Tablespaces mit System Managed Extent Size verwenden.

Für die meisten Datenbanken empfiehlt es sich, Smallfile-Tablespaces einzusetzen. Deren Flexibilität ist nicht nur höher, sondern Bigfile-Tablespace sind auch nur dann relevant, wenn die Datenbank aus Tausenden von Tablespaces besteht, oder die Größe mindestens einige zehn TB beträgt.

Praxistipp

Datafiles von Bigfile-Tablespaces sollten nur dann auf Raw-Devices und Filesystemen verwaltet werden, wenn man das Device oder das Filesystem erweitern kann. Dies ist beispielsweise der Fall, wenn ein Logical Volume Manager zum Einsatz kommt. Wird dies nicht beachtet und benötigt der Tablespace mehr Platz, als das Speichersystem zur Verfügung stellen kann, ist unter Umständen der gesamte Inhalt des Tablespace zu reorganisieren, beispielsweise durch Verschieben von Segmenten in einen anderen Tablespace.

4.4 Segmentspace-Verwaltung

Der vorangehende Abschnitt zeigt, wie die Database-Engine Extents alloziert und diese den Segmenten zuordnet. Mit anderen Worten, wie die Platzverwaltung auf Dateiebene funktioniert. In diesem Abschnitt gehen wir einen Schritt weiter und beschreiben die Platzverwaltung auf Segmentebene. Wir zeigen beispielsweise auf, wie Oracle die Rows bei einem Insert platziert, oder was mit dem freien Platz geschieht, wenn ein Datensatz gelöscht wird. Zu diesem Zweck verfügt Oracle über zwei Hauptstrategien: manuelle oder

automatische Segmentspace-Verwaltung. Erstere, bis einschließlich 8i die einzig verfügbare Methode, bedingt für eine optimale Funktion eine sorgfältige Konfiguration. Die zweite – neuer und daher von Oracle entsprechend favorisiert – erfordert für die korrekte Funktion keinerlei Konfiguration. Das Kapitel beschreibt nicht nur, wie diese beiden Strategien funktionieren, sondern erklärt auch, wie deren Vorteile von Nutzen sind. Zuerst wollen wir aber das Konzept der High-Water Mark kennenlernen.

4.4.1 High-Water Mark

Typischerweise sind nicht alle Blöcke eines Segmentes mit Daten belegt. Während einige Blöcke Daten enthalten oder einmal enthalten haben, sind andere Blöcke unbenutzt und deshalb unformatiert. Die Grenze zwischen den unformatierten und den formatierten Blöcken wird durch die sogenannte „High-Water Mark“ markiert. Per Definition liegen die formatierten Blöcke unterhalb dieser.

Die High-Water Mark spielt aus mehreren Gründen eine wichtige Rolle. Erstens hält sie die unformatierten Blöcke zusammen. Dies ist beispielsweise wichtig für ein Full-Scan auf ein Segment, der die Blöcke oberhalb der High-Water Mark nicht berücksichtigen muss; es sind lediglich die Blöcke unterhalb zu lesen. Dies ist besonders dann relevant, wenn das Verhältnis zwischen unformatierten und formatierten Blöcken groß ist. Zweitens werden bestimmte Operationen wie Direct-Inserts unterstützt. Um eine Direct-Insert Operation optimal auszuführen, darf die Database-Engine die bereits formatierten Blöcke unterhalb der High-Water Mark nicht berücksichtigen. Anstelle dessen bildet sie im Memory neue Blöcke, die direkt oberhalb der High-Water Mark in die Datafiles geschrieben werden.

Ein wichtiger Punkt ist, dass die High-Water Mark nicht heruntergesetzt wird, wenn man Daten aus einem Segment löscht. Die einzigen beiden Möglichkeiten, die High-Water Mark in diesem Fall herunterzusetzen, sind entweder das Segment neu zu erstellen (beispielsweise mit einem `ALTER TABLE MOVE`-Befehl), oder dessen Inhalt mit dem `ALTER TABLE SHRINK SPACE`-Befehl zu reorganisieren. Das folgende Beispiel zeigt den Effekt einer solchen Reorganisation. Beachten Sie, dass sich die Zahl der untersuchten Blöcke (buffer_gets) erst mit der Reorganisation ändert:

```
SQL> SELECT /* 1 */ count(*) FROM t;

  COUNT(*)
----------
     10000

SQL> SELECT sq.buffer_gets
  2  FROM v$session se, v$sql sq
  3  WHERE se.prev_sql_id = sq.sql_id
  4  AND se.prev_child_number = sq.child_number
  5  AND se.sid = sys_context('userenv','sid');

BUFFER_GETS
-----------
       1503
```

```
SQL> DELETE t WHERE id > 1000;

9000 rows deleted.

SQL> COMMIT;

Commit complete.

SQL> SELECT /* 2 */ count(*) FROM t;

  COUNT(*)
----------
      1000

SQL> SELECT sq.buffer_gets
  2  FROM v$session se, v$sql sq
  3  WHERE se.prev_sql_id = sq.sql_id
  4  AND se.prev_child_number = sq.child_number
  5  AND se.sid = sys_context('userenv','sid');

BUFFER_GETS
-----------
       1503

SQL> ALTER TABLE t SHRINK SPACE;

Table altered.

SQL> SELECT /* 3 */ count(*) FROM t;

  COUNT(*)
----------
      1000

SQL> SELECT sq.buffer_gets
  2  FROM v$session se, v$sql sq
  3  WHERE se.prev_sql_id = sq.sql_id
  4  AND se.prev_child_number = sq.child_number
  5  AND se.sid = sys_context('userenv','sid');

BUFFER_GETS
-----------
        220
```

Die Implementierung der High-Water Mark für die manuelle Platzverwaltung unterscheidet sich von der automatischen Segmentspace-Verwaltung. Die Beschreibung in diesem Abschnitt trifft mehr auf die manuelle Verwaltung zu. De facto unterhält Oracle bei der automatischen Segmentspace-Verwaltung zwei High-Water Marken: die sogenannte „Low-High-Water Mark" und die „High-High-Water Mark". Diese sind notwendig, weil keine exakte Grenze zwischen unformatierten und formatierten Blöcken existiert. Die Blöcke zwischen den beiden High-Water Marks können sowohl formatiert als auch unformatiert sein. Aus praktischer Sicht ist diese Unterscheidung jedoch nicht von Bedeutung und wir gehen deshalb an dieser Stelle nicht tiefer darauf ein. Tatsache ist jedoch, dass das Verhalten der High-High-Water Mark bei der automatischen Verwaltung konsistent mit der High-Water Mark bei der manuellen Segmentspace-Verwaltung ist.

4.4.2 Manuelle Segmentspace-Verwaltung

Die manuelle Segmentspace-Verwaltung basiert auf verlinkten Listen, die in einer Last-in-First-out (LIFO) Warteschlange stehen. Sinn und Zweck solcher Listen ist die Verwaltung des freien Platzes. Sie heißen deshalb auch „Freelist". Einfach gesagt ist eine Freelist eine verlinkte Liste, welche Datenblöcke referenziert, die freien Platz aufweisen. Demzufolge können nur Blöcke in dieser Liste stehen, die sich unterhalb der High-Water Mark befinden. Es gibt keinen spezifischen Ort, wo die gesamte Freelist gespeichert wird. Im Gegenteil, wie Abbildung 4.4 zeigt, kann eine Freelist über mehrere Blöcke verteilt sein. Der Headerblock des Segmentes beinhaltet neben diversen Basisinformationen eine Referenz zum ersten und zum letzten Block, der mit der Freelist verlinkt ist. Alle anderen Referenzen von Blöcken, die mit der Freelist verlinkt sind, werden – sofern vorhanden – im jeweiligen Blockheader gespeichert.

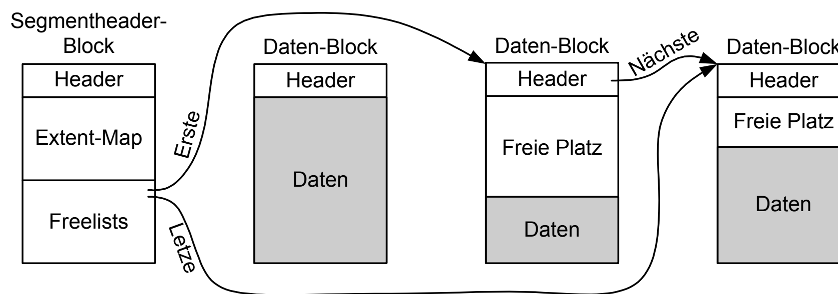

Abbildung 4.4 Segment mit vier Blöcken unterhalb der High-Water Mark: dem Segmentheader-Block, einen Datenblock ohne freien Platz und zwei teilweise gefüllte Datenblöcke, die auf der Freelist stehen

Nachdem wir nun die grundlegende Struktur einer Freelist beschrieben haben, wenden wir uns ihrer Verwaltung zu. Mit anderen Worten: Wann werden Datenblöcke mit einer Freelist verlinkt beziehungsweise wann wird diese Beziehung wieder aufgelöst?

Ein Prozess, der Daten in ein Tabellensegment einfügt, prüft zuerst, ob freier Platz in der Freelist vorhanden ist. Sobald ein Block mit dem erforderlichen freien Platz gefunden ist, werden die Daten eingefügt. Ist kein solcher Block vorhanden, werden die High-Water Mark erhöht und ein oder mehrere Blöcke mit einer Freelist verlinkt. Nachdem die Daten in den Block eingefügt sind, muss der Prozess prüfen, ob er den betroffenen Block wieder aus der Freelist entfernen muss. Die Auflösung der Verlinkung ist nur dann notwendig, wenn der freie Platz innerhalb des Blockes kleiner als `PCTFREE` ist. Es ist zu beachten, dass bei einem Indexsegment das Attribut `PCTFREE` nur bei der Erstellung relevant ist, danach wird es nicht mehr berücksichtigt. Indexblöcke werden verwendet, bis sie vollständig gefüllt sind, und falls mehr Platz benötigt wird, aufgeteilt.

Beim Löschen von Daten aus einem Tabellensegment muss der entsprechende Prozess prüfen, ob die betroffenen Blöcke mit einer Freelist verlinkt werden müssen oder nicht (vorausgesetzt die Blöcke sind nicht bereits auf einer Freelist). Eine Verknüpfung ist nur dann erforderlich, wenn der Block nicht bereits mit einer Freelist verlinkt ist und der freie

Platz größer als PCTUSED ist. Es ist zu beachten, dass PCTUSED für Indizes nicht spezifiziert werden kann, weil ein Indexblock nur dann mit einer Freelist verknüpft werden kann, wenn er leer ist.

Praxistipp

Die maximale Platzausnutzung lässt sich erzielen, wenn die Summe von PCTFREE und PCTUSED möglichst nahe bei 100 ist (die Summe kann 100 nicht überschreiten). Minimaler Aufwand bezüglich der Freelist-Verwaltung (Verknüpfungen von Blöcken erstellen und Verknüpfungen von Blöcken auflösen) lässt sich erreichen, indem die Summe von PCTFREE und PCTUSED möglichst klein gehalten wird (nahe bei 0). Dies bedeutet für die Konfiguration von PCTFREE und PCTUSED, dass man sich fragen muss, was wichtiger ist: maximale Platzausnützung in den Blöcken oder minimaler Verwaltungsaufwand für die Freelists. Generell ist es nicht möglich, beide Ziele gleichzeitig zu erreichen.

Standardmäßig hat ein Segment eine Freelist. Für Segmente, die sich selten ändern, ist das problemlos. In zwei Situationen kann jedoch eine einzelne Freelist zu Performance-Problemen führen. Erstens, wenn gleichzeitig mehrere Prozesse Daten in dasselbe Segment einfügen, ist es wahrscheinlich, dass sie zur gleichen Zeit auf den genau gleichen Block in der Freelist zugreifen (Zur Erinnerung: die Freelist wird sequentiell mit einem LIFO-Algorithmus verwaltet). Dies kann zu Engpässen führen und Verzögerungen bei der Datenblockmodifikation zur Folge haben. Zweitens werden die Strukturen im Headerblock des Segments modifiziert – beispielsweise Verknüpfungen von Blöcken erstellen und Verknüpfungen von Blöcken auflösen – können ebenfalls Engpässe auftreten. Diese sind in v$waitstat protokolliert. Es stehen zwei Methoden zur Lösung solcher Probleme zur Verfügung. Die erste Situation wird vermieden, indem man die Anzahl Freelists erhöht (FREELIST > 1). Die zweite Situation lässt sich vermeiden, indem die Strukturen, die zur Freelist gehören, aus dem Headerblock des Segmentes in sogenannte „Freelist-Blöcke“ verschoben werden, indem das Attribut FREELIST GROUPS größer als 1 gewählt wird. Es ist dabei zu beachten, dass das Attribut FREELISTS die Anzahl Freelists in jedem Freelist-Block definiert. Existieren mehrere Freelists und/oder Freelist-Blöcke, weist die Database-Engine jeden Prozess einer spezifischen Freelist zu. Dies geschieht, durch einen Hash-Algorithmus, der die Prozess-ID und – im Falle von Real Application Clusters – die Nummer der Instanz mit welcher der Prozess verbunden ist, als Parameter verwendet.

Das folgende Beispiel zeigt, wie die Freelist-Parameter beim Erstellen einer Tabelle spezifiziert werden:

```
SQL> CREATE TABLE t (n NUMBER)
  2  PCTFREE 25
  3  PCTUSED 25
  4  STORAGE (
  5    FREELISTS 4
  6    FREELIST GROUPS 2
  7  );
```

Praxistipp

Ein häufiges Missverständnis im Zusammenhang mit dem Attribut `FREELIST GROUPS` ist die Ansicht, dass dieses nur mit der Real-Application-Cluster-Option relevant ist. Aber das ist nicht korrekt, auch Single-Instanz-Datenbanken können von `FREELIST GROUPS` profitieren.

Folgende Einschränkungen betreffen die manuelle Freespace-Verwaltung:

- Das Verkleinern von Segmenten (mit dem Befehl `ALTER TABLE SHRINK SPACE`) mit dem Ziel, freien Platz unterhalb der High-Water Mark zu schaffen, ist nicht unterstützt.
- Der Segment Advisor unterstützt Segmente mit manueller Freespace-Verwaltung nicht.

4.4.3 Automatische Segmentspace-Verwaltung

Wie im vorangehenden Abschnitt erläutert, sind im Zusammenhang mit der Segmentspace-Verwaltung zwei Situationen zu verhindern: konkurrierender Zugriff auf Datenblöcke und konkurrierender Zugriff auf Headerblöcke. Dieser Abschnitt zeigt, wie die automatische Segmentspace-Verwaltung sicherstellt, dass diese beiden Probleme nicht auftreten.

Ein Segment mit automatischer Segmentspace-Verwaltung ist logisch in einer Baumstruktur organisiert, dessen Wurzel sich im Segmentheaderblock befindet. Die Verzweigungen sind durch Bitmapblöcke gebildet (es gibt zwei oder drei Ebenen). Am Ende der Baumstruktur befinden sich die Datenblöcke (leaf blocks). Abbildung 4.5 zeigt diese Struktur.

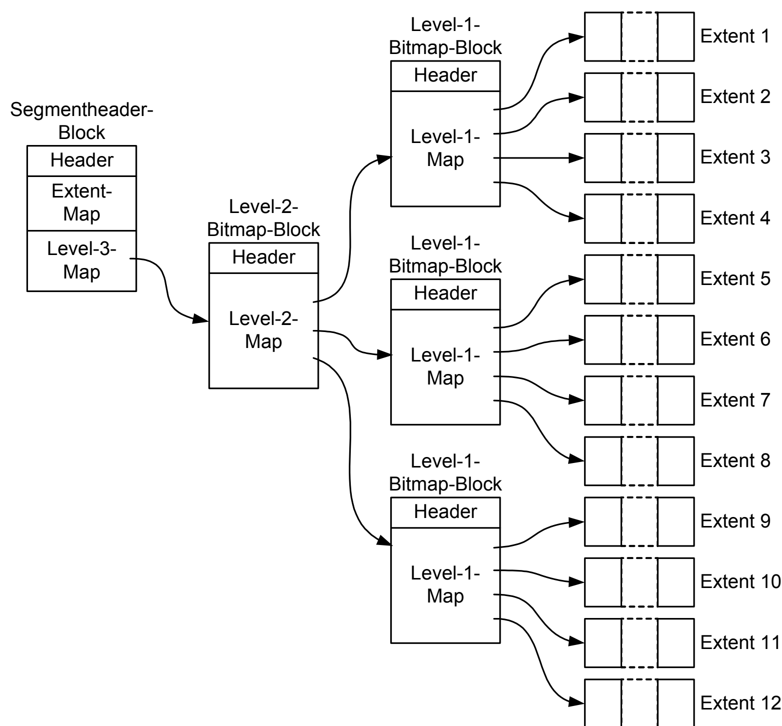

Abbildung 4.5 Ein Segment mit automatischer Segmentspace-Verwaltung ist logisch in einer Baumstruktur organisiert

Der Segmentheaderblock referenziert in der Level-3-Map auf Level-2-Bitmap-Blöcke, die mit dem Segment verknüpft sind. In den meisten Fällen ist diese Struktur sehr klein und passt vollständig in den Segmentheaderblock. Reicht der Platz nicht aus, wird ein Teil der Level-3-Map in anderen Blöcken gespeichert. Diese zusätzlichen Blöcke heißen Level-3-Bitmap-Blöcke (nicht erwähnt in Abbildung 4.5) und beinhalten nichts anderes, als Teile der Level-3-Map, die mit dem Segment verknüpft sind.

Die Level-2-Bitmap-Blöcke referenzieren in der Level-2-Map auf Level-1-Bitmap-Blöcke, mit denen sie verknüpft sind. Passt die Level-2-Map nicht vollständig in einem Block, werden zusätzliche Blöcke der Baumstruktur zugefügt. Für jeden Level-1-Bitmap-Block sind in der Level-2-Map zwei Informationen gespeichert: Erstens über den verfügbaren freien Platz der mit der entsprechenden Level-2-Map verknüpften Datenblöcke. Die zweite Information ist nur relevant, wenn mehrere Instanzen auf die Datenbank zugreifen. Sie weist aus, welche Instanz zu diesem bestimmten Level-1-Bitmap-Block eine Affinität hat. Über diese Affinität versucht die Database-Engine den Austausch von Level-1-Bitmap-Blöcken zwischen den Instanzen zu minimieren.

Der Level-1-Bitmap-Block referenziert in der Level-1-Map auf jedem Extent. Für jedes Extent sind darin auch detaillierte Informationen über den verfügbaren freien Platz in jedem Block vorhanden. Es ist zu beachten, dass alle Blöcke eines spezifischen Extents von einem einzigen Level-1-Bitmap-Block referenziert werden. Die maximale Anzahl Blöcke für einen Level-1-Bitmap-Block ist von der Segmentgröße abhängig:

- 16 für Segmente bis zu einer Größe von 1 MB
- 64 für Segmente bis zu einer Größe von 32 MB
- 256 für Segmente bis zu einer Größe von 1 GB
- 1024 für Segmente größer als 1 GB

Um eine zu häufige Aktualisierung der Bitmap-Blöcke zu verhindern, ist die Information über den freien Platz in den Bitmap-Blöcken mit geringer Präzision gespeichert. Statt dessen sind folgende Stati möglich:

- Der Block ist nicht formatiert.
- Der Block ist voll, oder er enthält Metadaten.
- Weniger als 25 Prozent des Blocks sind unbenutzt.
- Zwischen 25 Prozent (inklusiv) und 50 Prozent (exklusiv) des Blocks sind unbenutzt.
- Zwischen 50 Prozent (inklusiv) und 75 Prozent (exklusiv) des Blocks sind unbenutzt.
- Mindestens 75 Prozent des Blocks sind unbenutzt.

Nachdem wir nun gesehen haben, wie ein Segment logisch strukturiert ist, betrachten wir jetzt, wie diese Datenstrukturen für die Verwaltung des freien Platzes zur Anwendung kommen.

Ein Prozess, der Platz benötigt, um Daten in ein Segment einzufügen, durchläuft im Wesentlichen folgende Schritte:

- Zugriff auf den Segmentheader und Adresse des zuletzt verwendeten Level-2 Bitmap-Blocks ermitteln.
- Zugriff auf den Level-2-Bitmap-Block und Auswahl eines Level-1-Bitmap-Blocks, der Datenblöcke mit freiem Platz unterhalb der High-Water Mark referenziert. Ist kein solcher Block zu ermitteln, wird die High-Water Mark erhöht. Falls mehrere Level-1 Bitmap-Blöcke vorhanden sind, erfolgt die Auswahl aufgrund eines Hash-Algorithmus, der die Instanznummer und die Prozess-ID als Input verwendet. Das Ziel ist, konkurrierende Prozesse über mehrere Level-1-Bitmap-Blöcke zu verteilen.
- Zugriff auf den Level-1-Bitmap-Block und Auswahl eines Datenblocks mit freiem Platz. Sind mehrere Blöcke verfügbar, erfolgt die Auswahl aufgrund eines Hash-Algorithmus, der die Prozess-ID als Input verwendet. Das Ziel ist, konkurrierende Prozesse über mehrere Datenblöcke zu verteilen.
- Falls notwendig, wird die Freespace-Information im Bitmap-Block aktualisiert. Es ist zu beachten, dass der Block als gefüllt markiert wird, wenn der freie Platz kleiner als `PCTFREE` ist oder der Block für die Speicherung eines Index verwendet wird. Dies ist unabhängig davon, wie viel freier Platz im Block noch vorhanden ist.

Beim Löschen von Daten aus einem Segment ist zu prüfen, ob die Freespace-Information im Bitmap-Block zu aktualisieren ist. Beinhaltet der modifizierte Block einen Index, ist eine Aktualisierung nur notwendig, wenn der Block leer ist. Beinhaltet der modifizierte Block eine Tabelle, hängt die Entscheidung vom Status des Blocks vor der Löschoperation ab. Falls der Block vor der Löschoperation nicht als gefüllt markiert wurde, ist ein Update nur dann erforderlich, wenn man eine der oben beschriebenen Schwellwerte überschreitet. Beispielsweise ist ein Update dann erforderlich, wenn der Freespace von 10 auf 40 Prozent (der Schwellwert von 25 Prozent wurde überschritten) reduziert wurde, jedoch nicht, wenn der Freespace lediglich von 10 auf 20 Prozent gesunken ist. Falls der Block vor der Löschoperation als gefüllt markiert war, ist ein Update nicht nur dann erforderlich, wenn einer der oben beschriebenen Schwellwerte überschritten wurde, sondern auch, wenn der überschrittene Schwellwert größer als `PCTFREE` ist. Beispielsweise ist ein Update dann erforderlich, wenn `PCTFREE` auf 30 Prozent gesetzt wurde und der Freespace von 10 auf 60 Prozent (der Schwellwert von 50 Prozent wurde überschritten) reduziert wurde, jedoch nicht, wenn der Schwellwert von 10 auf 40 Prozent (lediglich der Schwellwert von 25 Prozent wurde überschritten) gesunken ist. Es ist zu beachten, dass `PCTUSED` nicht relevant ist für Segmente mit automatischer Segmentspace-Verwaltung.

Die folgenden Einschränkungen betreffen die automatische Segmentspace-Verwaltung:

- Diese Verwaltungsoption wird nur für permanente, locally-managed Tablespaces unterstützt.
- Diese Verwaltungsoption ist nicht für den SYSTEM-Tablespace unterstützt.

4.4.4 Auswahl einer Segmentspace-Verwaltungsoption

Weil in bestimmten Situationen die manuelle Segmentspace-Verwaltung nur bei korrekter Konfiguration gut funktioniert, empfehlen wir, wenn immer möglich, die automatische Verwaltungsmethode zu verwenden. Mit anderen Worten: Verwenden Sie die automatische Methode für alle permanenten, lokal verwalteten Tablespaces (mit Ausnahme des `SYSTEM`-Tablespace). Diese Wahl verhindert nicht nur Contention-Probleme, sondern vereinfacht auch die Reorganisation, um beispielsweise freien Platz unterhalb der High-Water Mark zurückzugewinnen. Dies gilt auch für Datenbanken oder einzelne Tablespaces, für die die Vorteile der automatischen Segmentspace-Verwaltung nicht relevant sind. So ist es beispielsweise sehr unwahrscheinlich, dass Tablespaces, die nur Indizes enthalten, oder eine typische ETL-Ladeoperation, aufgrund der Platzverwaltung Contention-Probleme haben.

Praxistipp

Verwenden Sie die manuelle Segmentspace-Verwaltung nur, wenn dies eine Anforderung ist, oder genügend Informationen und Zeit für die sorgfältige Konfiguration vorhanden ist.

Die automatische Segmentspace-Verwaltung hat zwei Nachteile: Erstens den Platzbedarf für die möglicherweise hohe Anzahl Bitmap-Blöcke. Solche Blöcke machen nicht nur die Segmente größer (der zusätzliche Platzbedarf beträgt 0,1 Prozent für große Segmente und bis 25 Prozent für sehr kleine Segmente), sie benötigen auch Platz im Buffercache (normalerweise wird weniger als 0,5 Prozent des Buffercaches für Bitmap-Blöcke verwendet, aber es ist nicht ungewöhnlich, wenn 1 bis 5 Prozent des Buffercaches für diesen Zweck erforderlich sind). Zweitens ist die Platzverwendung gegenüber der manuellen Methode aufgrund der ungenaueren Granularität der Freespace-Informationen nicht sonderlich effizient.

4.5 Zusätzliche Segmentoptionen

In diesem Abschnitt beschreiben wir zusätzliche Optionen, die auf Segmentebene möglich sind, im Speziellen die initiale Größe der Interested Transaction List und der Einsatz von Minimal Logging.

4.5.1 Interested Transaction List (ITL)

Jeder Datenblock beinhaltet eine Liste der aktiven Transaktionen auf diesem Block, die Interested Transaction List (ITL). Jeder Datensatz, der in eine Transaktion involviert ist, hat einen Pointer zur ITL. Einfach gesagt, wird dieser Pointer für das Locking auf Datensatzebene verwendet (siehe Abbildung 4.6). Jeder Eintrag in der ITL enthält neben anderen Informationen die Transaktions-ID und den Pointer zur Undo-Information dieser Transaktion.

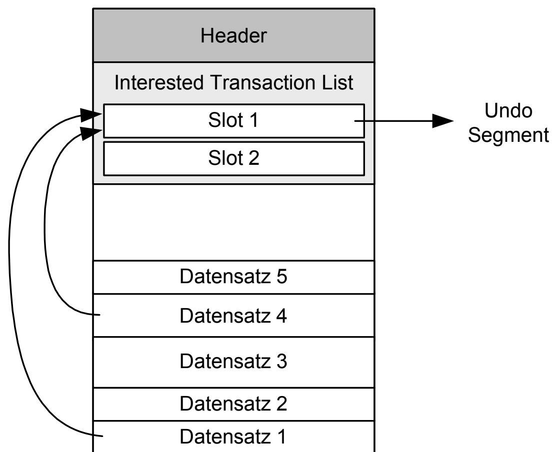

Abbildung 4.6
Die ITL enthält einige Slots, die für das Locking von modifizierten Datensätzen und für die Verknüpfung mit der entsprechenden Undo-Information verwendet werden. In der Abbildung hat die gleiche Transaktion Datensatz 1 und 4 geändert

Das Attribut `INITRANS` bestimmt die minimale Anzahl von Slots in der ITL. Auch wenn `INITRANS` auf 1 gesetzt sein kann, existieren immer mindestens zwei Slots. Vorausgesetzt, es ist genügend freier Platz vorhanden, lässt sich die Anzahl Slots bei Bedarf dynamisch erhöhen. Mit anderen Worten: Die Anzahl Slots steigt, wenn mehr aktive Transaktionen als Slots vorhanden sind. Ab 10g hängt die maximale Anzahl Slots von der Blockgröße ab und kann nicht mehr, wie in früheren Versionen, über den Parameter `MAXTRANS` spezifiziert werden.

Praxistipp

Es ist normalerweise besser, `PCTFREE` 1 bis 2 Prozent höher zu setzen, als einen hohen `INITRANS`-Wert zu definieren (jeder ITL-Slot benötigt 24 Bytes). Es gibt dafür zwei Gründe: Erstens kommt es nicht sehr häufig vor, dass viele Transaktionen gleichzeitig denselben Block modifizieren. Zweitens ist der von der ITL beanspruchte Platz bei Nichtgebrauch nicht für andere Zwecke wiederverwendbar.

Ist eine Transaktion nicht in der Lage, einen Slot zu erhalten oder einen neuen Slot zu kreieren (weil der benötigte freie Platz nicht verfügbar ist), muss sie warten, bis ein bestehender Slot freigegeben wird, sprich bis eine andere Transaktion ein Commit oder ein Rollback ausführt. Eine solche Situation nennt sich ITL-Wait. Es ist an dieser Stelle wichtig zu betonen, dass eine blockierte Session auf ein Shared Transaction Lock wartet. Deshalb kann man eine solche Situation mit der gleichen Methode untersuchen wie reguläre Lock-Contention. Die folgende Abfrage zeigt beispielsweise eine Session, die bereits zwölf Sekunden auf einen ITL-Slot gewartet hat. Beachten Sie, dass die Abfrage auch die den Lock verursachende Session (blocking session) zeigt:

```
SQL> SELECT event, seconds_in_wait, blocking_session
  2  FROM v$session
  3  WHERE sid = 147;

EVENT                        SECONDS_IN_WAIT BLOCKING_SESSION
---------------------------- --------------- ----------------
enq: TX - allocate ITL entry              12               24
```

Folgende Abfrage zeigt an, auf welchen Segmenten seit dem letzten Start der Datenbank ITL-Waits aufgetreten sind und wie oft dies vorgekommen ist:

```
SQL> SELECT owner, object_name, value
  2  FROM v$segment_statistics
  3  WHERE statistic_name = 'ITL waits'
  4  AND value > 0;
```

Es gibt drei Möglichkeiten, ITL-Waits zu verhindern beziehungsweise zu minimieren:

- Spezifizieren eines höheren `INITRANS`-Werts
- Verwendung eines höheren Werts für das `PCTFREE`-Attribut
- Verringerung der Anzahl gleichzeitiger Operationen auf einen einzelnen Datenblock. Treten ITL-Waits aufgrund von gleichzeitigen Updates auf, sollte man die Datendichte (Density) reduzieren (beispielsweise durch die Verwendung einer kleineren Blockgröße und einem höheren `PCTFREE`-Wert). Als Alternative kann auch eine andere Verteilung der Datensätze helfen (beispielsweise mittels Hash-Partitionierung). Treten ITL-Waits aufgrund gleichzeitiger Inserts auf und werden die Segmente manuelle verwaltet, dann sollte man die Anzahl der Freelists erhöhen.

Es ist zudem zu beachten, dass ITL-Waits zu ITL-Deadlocks führen können. Falls eine solche Situation auftritt, löst die Database-Engine typischerweise einen ORA-00060-Fehler aus (deadlock detected while waiting for resource).

4.5.2 Minimal Logging

Das Ziel von Minimal Logging ist die Minimierung der Redo-Erzeugung. Dies ist zwar eine optionale Einstellung, verbessert aber für bestimmte Operationen die Antwortzeit oft stark. Das Setzen des `NOLOGGING`-Attributs auf Segmentebene weist die Datenbank an, Minimal Logging zu verwenden. Es ist wichtig zu verstehen, dass Minimal Logging nur für Direct-Path Inserts und für bestimmte DDL-Anweisungen (beispielsweise `CREATE INDEX`, oder `CREATE TABLE AS SELECT`) unterstützt wird. Für alle anderen Operationen wird weiterhin die Redo-Information generiert.

Das folgende Beispiel zeigt die Auswirkung von Minimal Logging für eine `ALTER TABLE MOVE`-Operation. Es ist zu beachten, dass die im Beispiel verwendete Tabelle ca. 1 MB Daten enthält und eine Move-Operation deshalb auch zu etwa 1 MB Redo-Information führen sollte. Wie das Beispiel zeigt, wird mit Minimal Logging wesentlich weniger Redo-Information generiert als mit normalem Logging:

```
SQL> ALTER TABLE t NOLOGGING;

SQL> SELECT value
  2  FROM v$mystat NATURAL JOIN v$statname
  3  WHERE name = 'redo size';

     VALUE
----------
      6152

SQL> ALTER TABLE t MOVE;
```

```
SQL> SELECT value
  2  FROM v$mystat NATURAL JOIN v$statname
  3  WHERE name = 'redo size';

     VALUE
----------
     58764

SQL> ALTER TABLE t LOGGING;

SQL> ALTER TABLE t MOVE;

SQL> SELECT value
  2  FROM v$mystat NATURAL JOIN v$statname
  3  WHERE name = 'redo size';

     VALUE
----------
   1291940
```

Praxistipp

Wir empfehlen, Minimal Logging nur dann zu verwenden, wenn die resultierenden Auswirkungen vollständig klar sind. So lassen sich beispielsweise Datenblöcke, die man mit Minimal Logging modifiziert hat, nicht mittels Media-Recovery wiederherstellen. Dies bedeutet, dass die Database-Engine nach einem Media-Recovery diese Blöcke lediglich als „logical corrupted“ bezeichnen kann, weil das Media-Recovery für die Rekonstruktion des Blockinhalts Zugriff auf die entsprechende Redo-Information benötigt. Dies führt dazu, dass SQL-Anweisungen, die auf Objekte mit solchen Blöcken zugreifen, mit einem ORA-26040-Fehler (Data block was loaded using the NOLOGGING option) abbrechen. Deshalb sollte man Minimal Logging nur dann verwenden, wenn die Daten manuell wieder geladen werden können oder wenn unmittelbar nach der Ladeoperation ein Backup erfolgt.

Folgende Besonderheiten betreffen den Einsatz von Minimal Logging:

- Es ist möglich, auf Datenbank- oder Tablespace-Ebene `FORCE LOGGING` zu aktivieren und damit die Einstellungen auf Segmentebene zu übersteuern. Dies ist speziell für Standby-Datenbanken und Streams nützlich. In der Tat funktionieren diese beiden Optionen nur dann erfolgreich, wenn die Redolog-Files Informationen von allen Modifikationen enthalten.
- Minimal Logging wird für Segmente, die in temporären Tablespace mit Tempfiles abgelegt sind, immer genutzt.
- Minimal Logging kommt für Datenbanken, die im `NOARCHIVELOG`-Mode betrieben werden, immer vor.

4.6 Reorganisationen

Es gibt drei Gründe, um ein Segment zu reorganisieren:

- Man will das Segment in einen anderen Tablespace verschieben.
- Man will den freien Platz unterhalb der High-Water Mark zurückgewinnen. Im Allgemeinen wird der freie Platz von der Database-Engine automatisch wiederverwendet und eine Reorganisation sollte nicht notwendig sein. Ausnahme, wenn viele Datensätze gelöscht werden (es kann dabei lange dauern, bis der freie Platz wiederverwendet wird), oder wenn man neue Datensätze mittels Direct-Inserts einfügt.
- Man will migrierte oder verkettete Datensätze eliminieren.

Bevor wir die verschiedenen Reorganisationsmethoden beschreiben, ist es sinnvoll, den Unterschied zwischen Datensatzmigration (row migration) und Datensatzverkettung (row chaining) zu betrachten.

4.6.1 Datensatzmigration und Datensatzverkettung

Datensatzmigration und Datensatzverkettung wird oft verwechselt. Dies hat aus unserer Sicht zwei Hauptgründe: Erstens lassen sich die beiden Situationen leicht verwechseln, weil sie bestimmte gemeinsame Eigenschaften aufweisen. Zweitens wurden die beiden Begriffe in der Oracle-Dokumentation und -Implementation nie sehr konsistent unterschieden.

Beim Einfügen von Datensätzen in einen Block reserviert die Database-Engine freien Platz für zukünftige Updates. Wie in diesem Kapitel bereits beschrieben, wird der für Updates reservierte Platz über das Attribut `PCTFREE` festgelegt. Abbildung 4.7 zeigt, dass ein Block, dessen Füllgrad `PCTFREE` erreicht hat, für Inserts nicht mehr zur Verfügung steht.

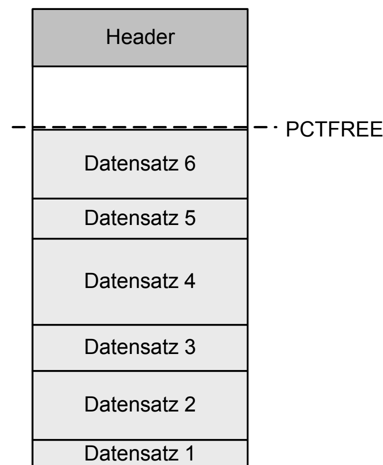

Abbildung 4.7
Bei Inserts wird im Datenblock Platz (PCTFREE) für zukünftige Updates freigehalten.

Wird ein Datensatz aktualisiert und beansprucht dadurch mehr Platz, sucht die Database-Engine freien Platz im gleichen Block. Ist zu wenig Platz vorhanden, teilt sie den Datensatz in zwei Teile auf. Der erste Teil (der lediglich Kontrollinformationen wie die Row-Id zum zweiten Teil enthält) verbleibt im Originalblock. Damit wird eine Änderung der Row-Id verhindert, denn dies darf auf keinen Fall vorkommen, weil die Row-Id nicht nur in den Indizes permanent gespeichert wird, sondern auch temporär im Memory. Der zweite Teil, der die Daten enthält, wird in einem anderen Block abgelegt. Bei dem geänderten Datensatz spricht man von einem migrierten Datensatz. In Abbildung 4.8 sieht man beispielsweise, dass Datensatz Nummer 4 in einen zweiten Block migriert wurde.

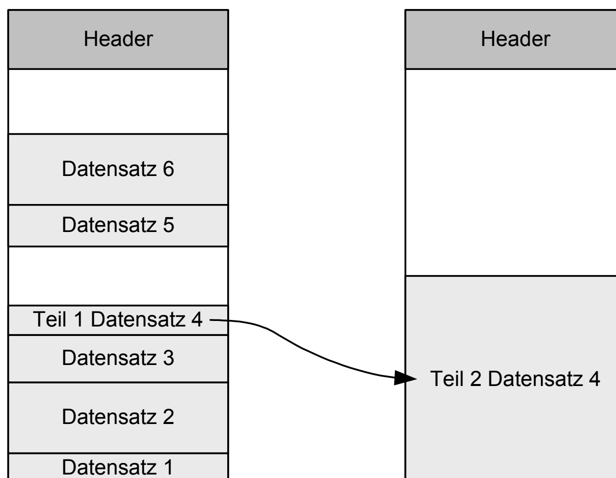

Abbildung 4.8
Ein aktualisierter Datensatz, der keinen Platz mehr im Originalblock findet, wird in einen anderen Block migriert (Datensatzmigration)

Passt ein Datensatz aufgrund seiner Größe nicht in einen einzelnen Datenblock, wird er ebenfalls in zwei, oder mehrere Stücke aufgeteilt. Dann ist jedes Stück in einem separaten Block gespeichert und es wird eine Verknüpfung beziehungsweise Verkettung zwischen den einzelnen Teilen gebildet. Einen solchermaßen geänderten Datensatz bezeichnet man als verketteten Datensatz (Chained Row*)*. Abbildung 4.9 (nächste Seite) zeigt, wie ein Datensatz über drei Blöcke hinweg verkettet ist.

Tabellen mit vielen Attributen können ebenfalls zu Datensatzverkettung führen, weil die Database-Engine nicht in der Lage ist, mehr als 255 Spalten pro Datensatz-Teil zu speichern. Konsequenterweise muss ein solcher Datensatz in mehrere Stücke aufgeteilt sein. Diese Situation ist insbesondere dann speziell, wenn die Stücke, die zum gleichen Datensatz gehören, im selben Block gespeichert werden. Dies nennt sich dann „Intra-Block Chaining".

Datenaktualisierung verursacht Datensatzmigration, während Datensatzverkettung entweder durch Einfügen oder durch Datenaktualisierung entsteht. Ist Datenaktualisierung die Ursache für Datensatzverkettung, werden die Datensätze gleichzeitig migriert und verkettet.

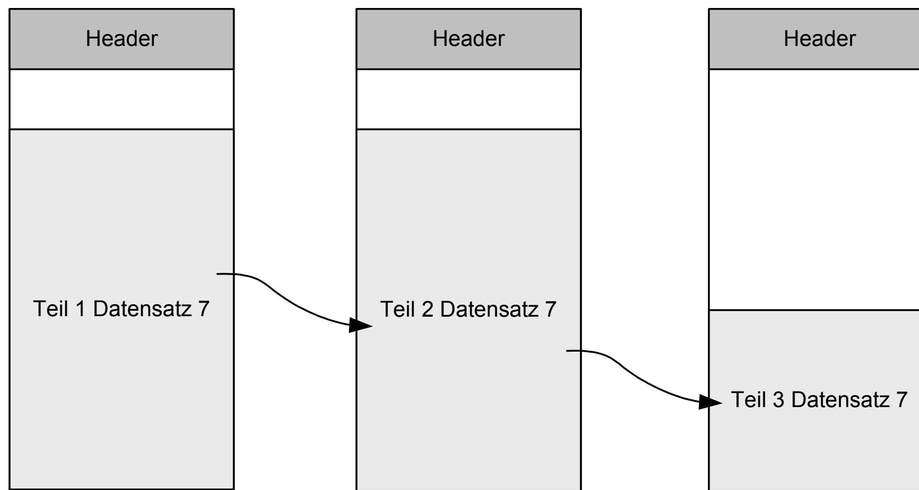

Abbildung 4.9 Ein verketteter Datensatz ist über zwei oder mehrere Blöcke verteilt

Der Performance-Einfluss von Datensatzmigration ist vom Zugriffspfad auf die Datensätze abhängig. Erfolgt der Zugriff via RowId verdoppelt sich der Ressourcenbedarf, weil der Zugriff auf beide Teile des Datensatzes separat erfolgt. Andererseits gibt es keinen Nachteil, wenn der Zugriff über einen Full-Scan erfolgt, weil der erste Teil des Datensatzes, der keine Daten enthält, einfach übersprungen wird.

Der Einfluss von Datensatzverkettung auf die Performance ist unabhängig vom Zugriffspfad. Jedes Mal, wenn der erste Teil eines Datensatzes gefunden wird, müssen alle anderen Teile über die RowId gelesen werden. Es gibt eine Ausnahme: Ist nur ein Teil des Datensatzes erforderlich, müssen nicht alle Teile gelesen werden. Werden beispielsweise nur Attribute aus dem ersten Teilstück benötigt, braucht man die anderen Teile des Datensatzes nicht zu lesen.

Es gibt zwei wichtige Methoden, um Datensatzmigration und Datensatzverkettung zu identifizieren. Die erste basiert auf den beiden Views `V$SYSSTAT` und `V$SESSTAT`, die Hinweise auf Datensatzmigration und Datensatzverkettung in der Datenbank beinhalten. Die Idee dahinter ist, die Systemstatistik „table fetch continued row" zu prüfen, um die Anzahl Zugriffe auf Datensätze mit mehr als einem Stück (inkl. Intra-Block Chained Rows) auszuweisen. Die relative Auswirkung von Datensatzverkettung und Datensatzmigration lässt sich auch mit dem Vergleich der beiden Statistiken „table scan rows gotten" und „table fetch by rowid" beurteilen. Im Gegensatz dazu erhält man mit der zweiten Methode exakte Informationen über die migrierten und die verkettete Datensätze. Dies bedingt jedoch die vorgängige Analyse jeder Tabelle, die unter Verdacht steht, verkettete oder migrierte Datensätze zu enthalten, mit der SQL-Anweisung `ANALYZE TABLE LIST CHAINED ROWS`. Die RowId für verkettete oder migrierte Rows ist in der Tabelle `CHAINED_ROWS` eingetragen. Wie die folgende Abfrage zeigt, kann man danach, basierend auf den Row-Ids, die Größe der Datensätze und daraus – durch Vergleich mit der Blockgröße – Datensatzmigration oder Datensatzverkettung erkennen:

```
SELECT vsize(<col1>) + vsize(<col2>) + ... + vsize(<coln>)
FROM <table>
WHERE rowid = '<rowid>'
```

Praxistipp

Das Attribut `CHAIN_CNT` der `DBA_TABLES`-View sollte die Anzahl verkettete und migrierte Rows beinhalten. Leider ermittelt das DBMS_STATS-Package diese Statistik nicht. Der Wert ist einfach auf „0" gesetzt. Demnach ist die Ausführung des SQL-Befehls `ANALYZE TABLE COMPUTE STATISTICS` der einzige Weg, korrekte Statistikwerte zu berechnen. Dies führt allerdings zu einem unerwünschten Nebeneffekt, da hinterher die Objektstatistiken der analysierten Tabellen überschrieben sind.

Die Maßnahmen für die Verhinderung von Datensatzmigration und Datensatzverkettung unterscheiden sich grundsätzlich. Es ist deshalb auch zu unterscheiden, ob Probleme durch Datensatzmigration oder durch Datensatzverkettung entstanden sind.

Das Verhindern von Datensatzmigration ist lediglich eine Frage der korrekten `PCTFREE`-Einstellung. Mit anderen Worten: Bei einer ausreichenden Platzreservierung findet ein modifizierter Datensatz vollständig im Originalblock Platz. Stellen Sie also Datensatzmigration fest, sollten Sie den aktuellen `PCTFREE`-Wert erhöhen. Für die Wahl eines geeigneten `PCTFREE`-Werts ist das durchschnittliche Datensatzwachstum abzuschätzen. Dies gelingt am besten, wenn Sie die durchschnittliche Datensatzgröße vergleichen, unmittelbar nach dem Einfügen und nachdem keine Änderung mehr erfolgt.

Die Verhinderung von Datensatzverkettung ist schwieriger. Die offensichtlichste Maßnahme ist die Verwendung von größeren Datenbankblöcken. Doch manchmal ist selbst die größtmögliche Blockgröße nicht ausreichend. Tritt Datensatzverkettung aufgrund von mehr als 255 Attributen auf, kann nur ein Re-Design der Tabelle Abhilfe schaffen. Deshalb ist die einzige wirksame Maßnahme in dieser Situation, die weniger oft verwendeten Spalten am Ende der Tabelle zu platzieren und somit den gleichzeitigen und übergreifenden Zugriff auf alle Teile des Datensatzes zu verhindern.

4.6.2 Verschieben von Segmenten

Die Idee dieser Methode ist das vollständige Verschieben eines Segments von der aktuellen Lokation in eine andere. Daher ist diese Methode für die Verschiebung eines Segments in einen anderen Tablespace, für die Freigabe von freiem Platz unterhalb der High-Water Mark oder für die Eliminierung von migrierten Rows möglich. Mit dem Verschieben des Segments in einen Tablespace mit größerer Blockgröße lässt sich auch Datensatzverkettung eliminieren.

Der Segmenttyp bestimmt die SQL-Anweisung, die für die Verschiebung Anwendung findet. Die folgenden Beispiele zeigen einige gebräuchliche SQL-Anweisungen;

- Tabellensegment (inklusiv Segmente von Index-Organized Tabellen) verschieben:

```
SQL> ALTER TABLE t MOVE TABLESPACE users;
```

- Table-Partition-Segment (inklusiv Segmente von partitionierten, Index-Organized Tabellen) verschieben (P1 ist der Name der Partition):
  ```
  SQL> ALTER TABLE t MOVE PARTITION p1 TABLESPACE users;
  ```
- Indexsegment verschieben:
  ```
  SQL> ALTER INDEX i REBUILD TABLESPACE users;
  ```
- Index-Partition-Segment verschieben (P1 ist der Name der Partition):
  ```
  SQL> ALTER INDEX i REBUILD PARTITION p1 TABLESPACE users;
  ```
- LOB-Segment verschieben (B ist der Name der LOB):
  ```
  SQL> ALTER TABLE t MOVE LOB (b) STORE AS (TABLESPACE users);
  ```

Alle Beispiele bedingen exklusiven Zugriff auf das zu reorganisierende Segment. Es gibt nur einen Segmenttyp, der eine Online-Reorganisation direkt unterstützt, Indexsegmente (inklusiv Segmente von Index-Organized-Tabellen). Eine solche Online-Reorganisation (Oracle Enterprise Edition erforderlich) muss mit dem Schlüsselwort ONLINE ausgeführt werden, wie das folgende Beispiel zeigt:

```
SQL> ALTER INDEX i REBUILD ONLINE TABLESPACE users;
```

Ist eine Online-Reorganisation für einen anderen Segmenttyp notwendig, bietet sich das `DBMS_REDEFINITION`-Package an, obwohl der Hauptzweck dieses Package in der Änderung von Tabellenstrukturen liegt.

4.6.3 Verschieben von Tabelleninhalten

Besteht das Reorganisationsziel darin, Datensatzmigration zu eliminieren, ist die vollständige Reorganisation, wie sie im vorangehenden Abschnitt beschrieben wurde, nicht in jedem Fall die beste Lösung. Insbesondere wenn das Segment groß und die Zahl der migrierten Datensätze relativ klein sind (einige Prozent). In einer solchen Situation ist es eventuell sinnvoller, nur die migrierten Rows zu verschieben. Diese Methode funktioniert jedoch nicht zur Beseitigung von Datensatzverkettung.

Die folgende Beschreibung zeigt, wie man die migrierten Datensätze einer Tabelle T verschiebt;

- Lokalisieren der migrierten Rows (Dieser Schritt lokalisiert eigentlich migrierte und verkettete Datensätze). Beachten Sie, dass dieser Schritt weder exklusiven Zugriff auf die zu analysierende Tabelle bedingt, noch Wartezeit auf offene Transaktionen verursacht:
  ```
  SQL> @?/rdbms/admin/utlchain.sql
  SQL> ANALYZE TABLE t LIST CHAINED ROWS;
  ```
- Erzeugen einer Scratch-Tabelle, die die gleiche Struktur aufweist wie die zu reorganisierende Tabelle:
  ```
  SQL> CREATE TABLE t_scratch AS
    2  SELECT *
    3  FROM t
    4  WHERE 1 = 0;
  ```

- Sperren der migrierten (und verketteten) Rows. Dieser Schritt ist nur dann notwendig, wenn man die Reorganisation ohne exklusiven Zugriff auf die zu reorganisierende Tabelle durchführt:

```
SQL> SELECT *
  2  FROM t
  3  WHERE rowid IN (SELECT head_rowid FROM chained_rows)
  4  FOR UPDATE;
```

- Eintragen (Insert) der migrierten (und verketteten) Rows in die Scratch-Tabelle:

```
SQL> INSERT INTO t_scratch
  2  SELECT *
  3  FROM t
  4  WHERE rowid IN (SELECT head_rowid FROM chained_rows);
```

- Löschen der migrierten (und verketteten) Rows aus der Originaltabelle:

```
SQL> DELETE t
  2  WHERE rowid IN (SELECT head_rowid FROM chained_rows);
```

- Wiedereinfügen der migrierten (und verketteten) Rows in die Originaltabelle:

```
SQL> INSERT INTO t
  2  SELECT * FROM t_scratch;
```

- Committen der Änderungen:

```
SQL> COMMIT;
```

- Löschen der Scratch-Tabelle:

```
SQL> DROP TABLE t_scratch PURGE;
```

Soll die Reorganisation online erfolgen und sind viele Datensätze zu verschieben, kann die Verarbeitung auch Mengenweise (beispielsweise 1000 Datensätze auf einmal) geschehen. Damit erfolgen anstelle einer einzigen langen Transaktion einige kleine Transaktionen.

Weil Datensätze bei dieser Reorganisationsmethode gelöscht und eingefügt werden, erfordern Trigger und Fremdschlüssel auf die zu reorganisierenden Tabellen spezielle Aufmerksamkeit. Existieren solche Objekte, ist ein exklusiver Zugriff auf die betroffenen Tabellen und die Deaktivierung von Triggern und Fremdschlüssel erforderlich.

4.6.4 Rückgewinnung von freiem Platz

Für die Rückgewinnung von freiem Platz unterhalb der High-Water Mark ist die vollständige Verschiebung des Segmentes in eine andere Lokation nicht zwingend notwendig. Vor allem in zwei Situationen möchte man dies verhindern. Erstens, wenn der freie Platz im Verhältnis zur Segmentgröße sehr klein ist, und zweitens, wenn nicht genügend Platz vorhanden ist, um das Segment zweifach zu speichern.

Ist das zu reorganisierende Segment mit einem Index verbunden, kann man die `COALESCE`-Klausel im `ALTER INDEX`-Befehl verwenden. Beachten Sie, dass diese Methode die High-Water Mark nicht verringert. Sie fügt hingegen den Inhalt der benachbarten Indexblöcke zusammen und löst die Verkettung der freien Blöcke von den Indexstrukturen. Beachten Sie zudem, dass kein exklusiver Zugriff auf den zu reorganisierenden Index notwendig ist.

Im folgenden Beispiel zeigen wir, wie der Befehl auf einen nicht-partitionierten Index angewendet wird. Für einen partitionierten Index sollte zusätzlich die `PARTITION`-Klausel spezifiziert sein.

```
SQL> ALTER INDEX i COALESCE;
```

Ist das zu reorganisierende Segment mit einer Tabelle verbunden, ist die `SHRINK`-Klausel im `ALTER TABLE`-Befehl möglich. Das Ziel dieser Segment-Shrink-Methode ist, so viele Datensätze wie möglich am Anfang des Segmentes zu speichern. Zu diesem Zweck werden die Datensätze verschoben und erhalten damit eine neue Row-ID. Beachten Sie zudem, dass kein exklusiver Zugriff auf das zu reorganisierenden Segment notwendig ist. Voraussetzung ist jedoch die vorgängige Aktivierung von Row-Movement, wie das folgende Beispiel zeigt:

```
SQL> ALTER TABLE t ENABLE ROW MOVEMENT;
SQL> ALTER TABLE t SHRINK SPACE;
```

Standardmäßig sind mit Segment-Shrink die Datensätze nicht nur am Anfang des Segments gespeichert, sondern es werden auch die High-Water Mark verringert und die meisten freien Blöcke unterhalb der High-Water Mark freigegeben. Ist die Verringerung der High-Water Mark nicht erforderlich, kann man auch die `COMPACT`-Klausel verwenden:

```
SQL> ALTER TABLE t SHRINK SPACE COMPACT;
```

Eine weitere Option, die während der Shrink-Operation spezifiziert werden kann, ist `CASCADE`. Damit werden nicht nur die Tabelle, sondern auch alle abhängigen Segmente wie Indexe und LOBs verarbeitet. Zudem kann man auch eine einzelne Partition, oder nur ein abhängiges Segment verarbeiten, wie das folgende Beispiel zeigt;

- Shrink einer einzelnen Partition (P1 ist der Name der Partition):

```
SQL> ALTER TABLE t MODIFY PARTITION p1 SHRINK SPACE;
```

- Shrink eines LOB-Segmentes (B ist der Name des LOB):

```
SQL> ALTER TABLE t MODIFY LOB (b) (SHRINK SPACE);
```

In folgenden Fällen wird Segment-Shrink nicht unterstützt:

- Für Tabellen mit manueller Segmentspace-Verwaltung (also für Tabellen, die nicht in einem ASSM-Tablespace gespeichert sind)
- Für Tabellen mit Function-Based-Indizes, Domain-Indizes, oder Bitmap-Join-Indizes
- Für komprimierte Tabellen
- Für Tabellen mit LONG-Attributen
- Für Tabellen-Cluster

4.7 Resümee

In diesem Kapitel haben wir beschrieben, wie Oracle die Daten auf logischer und physischer Ebene speichert und verwaltet. Für diesen Zweck haben wir nicht nur die Möglichkeiten aufgezeigt, die uns die Datenbank-Engine zur Verfügung stellt, sondern auch die Eigenschaften beschrieben, die man von einem Speichersubsystem erwarten darf. Weil beinahe auf jeder Ebene mehrere Optionen für die gleiche Aufgabe existieren, legen wir ein spezielles Augenmerk auf die richtige, sprich situationsgerechte Auswahl. Wir haben zudem aufgezeigt, welche Reorganisationsmethoden angewendet werden können, wenn die Daten nicht optimal gespeichert sind.