Oracle 7/8/9/10 - Datenbankadministration

Hinweis: Ergänzungen zur Version 9/10 erfolgen an erforderlicher Stelle.

1 Aufgaben eines DBA....3
2. ORACLE 7/8/9 Systemüberblick....4
2.1 Übersicht....4
2.1.1 Datenfluss in einem Datenbanksystem....4
2.1.2 Aufbau von Oracle - Überblick....4
2.2 Aufbau der SGA (Shared Memory)....6
2.3 Server- und Benutzerprozesse....10
2.5 SQL-Anweisungen....12
2.5.1 Verarbeitung von SQL-Anweisungen....12
2.5.2 Schritte bei SQL-Anweisungen....13
2.5.3 SELECT-Anweisung....13
2.5.4 UPDATE-Anweisung....14
2.6 Der Data-Base-Writer-Prozess (DBWR)....17
2.7 Transaktionen....17
2.7.1 Einführung....17
2.7.2 Transaktionen bei Oracle....18
2.7.3 Der Log Writer-Prozess (LGWR)....19
2.7.4 Datenbank-Commits....20
2.7.5 Redo Log-Dateien....20
2.8 Checkpoints....22
2.9 Die Hintergrundprozesse PMON, SMON und ARCH....24
2.10 Die Prozesse RECO, LCKn, Pnnn und SNPn....26
2.11 Parameterdatei....27
2.12 Trace-Dateien und Alert-Dateien....27
3 Sperrmodell....29
3.1 Lesekonsistenz....29
3.2 Multiversion Concurrency Control System....30
3.3 Die Auswahl eines Isolationlevels....31
3.4 Explizite Sperren....35
3.4.1 Übersicht....35
3.4.2 DML Locks....35
3.4.3 Default Locking für Abfragen....41
3.4.4 DDL Locks (Dictionary Locks)....42
4. Instance....47
5. Datenbankadministration....49
5.1 Datenbank erzeugen (Übersicht)....49
5.1.1 Datenbankstruktur von Oracle 8i....49
5.1.2 Abbildung der logische Db-Struktur auf die physische Struktur....51
5.2 Übersicht über das Erzeugen und Löschen einer Datenbank....57
5.2.1 Erzeugen einer Datenbank....57
5.2.2 Löschen einer Datenbank....67
5.3 Datenbankstruktur verwalten....68
5.3.1 Tablespaces....68
5.3.2 Richtlinien für die Verwaltung von Tablespaces....74
5.4 Segmente....76
5.5 Rollbacksegmente verwalten....78
6. Tabellen-, Index- und Clustersegmente....83
6.1 Tabellen verwalten....83
6.2 Indizes....89
6.2.1 B*-Bäume....92
6.2.2 Bitmap-Indizierung....94
6.3 Constraints....96

6.4 Clustersegmente verwalten 104
7 Benutzer verwalten 108
8 Ressourcenverwaltung (Benutzerprofile) 110
9 Datenbankzugriff verwalten 113
9.1 Datenbankprivilegien - Einführung 113
9.2 Rechte vergeben und entziehen 116
9.3 Rollen - Benutzerklassen und Zugriffskontrolle 117
10 Datenbankaudit 119
11 Serverkonfigurationen 122
11.1 Allgemeines zur Serverkonfiguration 122
11.2 Oracle Netzwerk 136

1 Aufgaben eines DBA

Datenbank installieren
Datenbank hoch und herunterfahren
Benutzer eintragen und überwachen
Datenbankprivilegien vergeben (Systemsicherheit)
Datenbankspeicher planen und verwalten
Datenbank sichern und wiederherstellen
mit Audit Statistiken erstellen
Datenbank entwerfen und anlegen

Wichtig bei verteilten DB: NLS (National Language Support)

Bei größeren DB-Systemen gibt es mehrere DBA s mit unterschiedlichen Rechten.

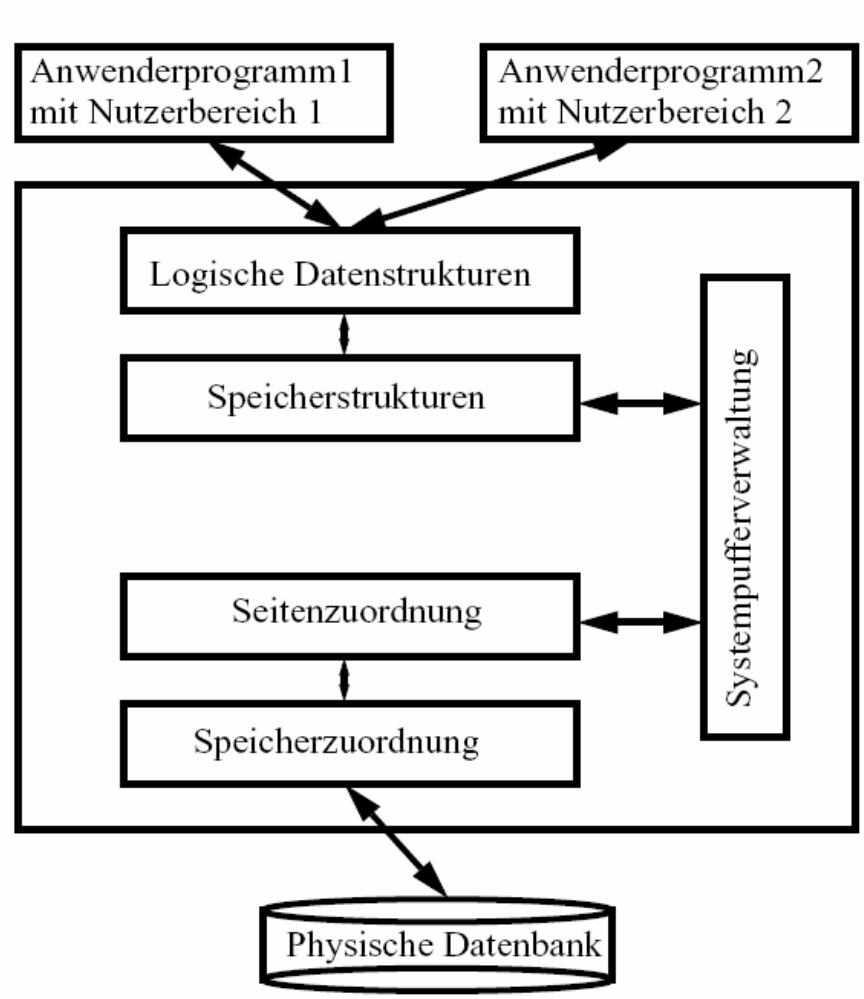
Anwendungsentwickler planen und entwickeln Anwendungen, Datenobjekte und tunen sie.

2. ORACLE 7 / 8 / 9 Systemüberblick

2.1 Übersicht

2.1.1 Datenfluss in einem Datenbanksystem

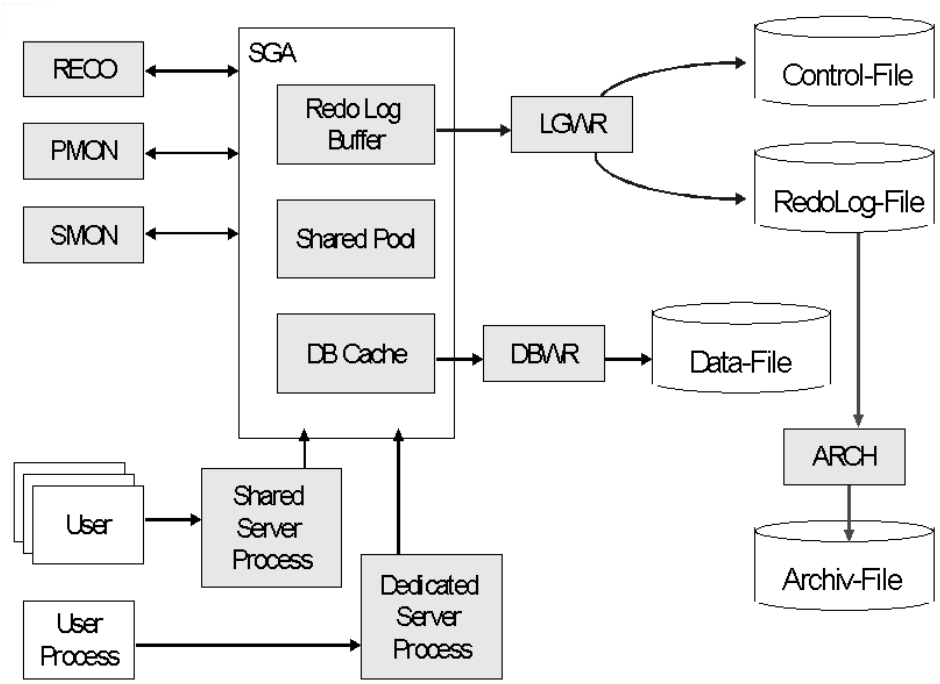
Hinweis: Dieser Aufbau entspricht der 3-Schichten-Architektur (ANSI-Sparc)



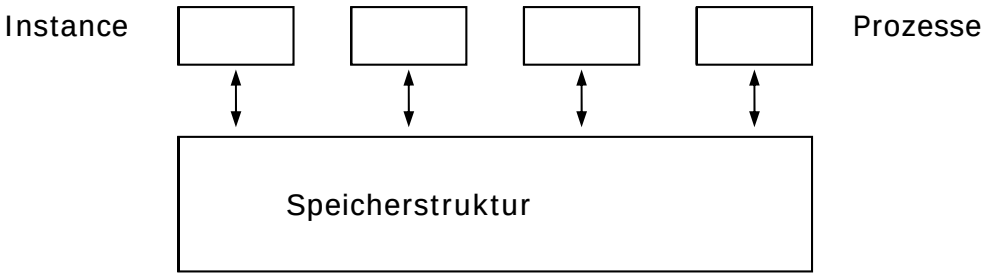
2.1.2 Aufbau von Oracle - Überblick

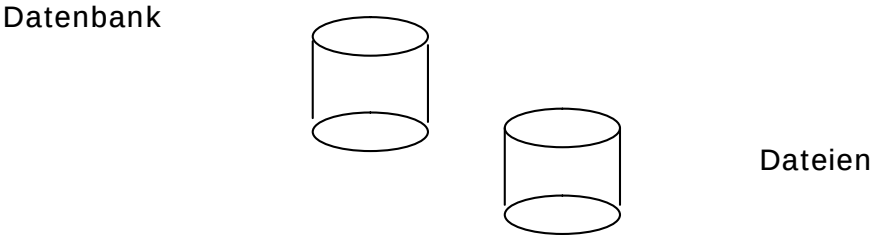
Der Systemüberblick behandelt im Wesentlichen
- Prozesse
- Speicherstrukturen
- Dateien

Systemüberblick



Oracle Datenbanksystem (vereinfacht)



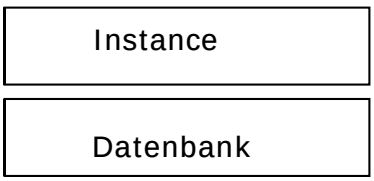


Instance: aktiver Teile des DBS. Führt Aktivitäten aus, die Benutzerprozess nicht ausführen kann oder soll. Besteht aus SGA (System Global Area) und Hintergrundprozesse.

Hinweis: Pro Database üblicherweise eine Instanz. Ausnahme: Mehrere Instanzen pro Database bei verteilten Systemen.

Datenbank: Dies ist die eigentliche Datenbasis, die in Form von Dateien vorliegt. Sie ist passiv.

Eine Instance wird gestartet und mit einer Datenbank verbunden (mount).



Datenbanksystem

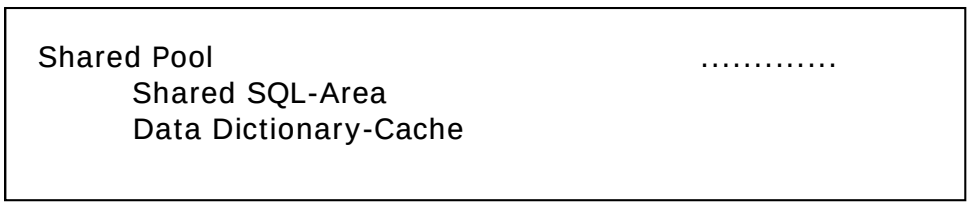
Name der Instance: ORACLE System Identifier (SID)
Name der Datenbank: DB_NAME (Parameter der init.ora-Datei)

2.2 Aufbau der SGA (Shared Memory)

Hinweis: Kontrolle unter Linux: ipcs m
SQLPLUS: show sga

Shared Pool

System Global Area



Shared SQL-Area

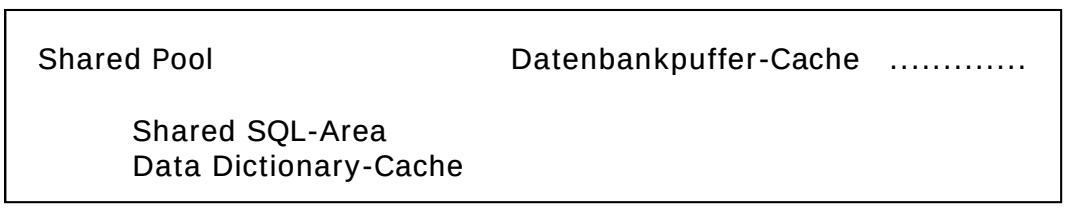
Enthält alle SQL-Anweisungen (von allen Benutzern der DB). Diese werden nur einmal abgelegt, wenn diese vollständig identisch sind (nur durch genaue Vorschriften der Schreibweise möglich)

Data Dictionary-Cache

Enthält Namen aller Tabellen und View s
Namen und Datentypen der Spalten der Datenbank-Tabellen
Privilegien allen Oracle-Benutzer

Datenbankpuffer-Cache

System Global Area



Datenbankpuffer-Cache: Dieser enthält Datenblöcke (Seiten), die von der Platte gelesen wurden. Alle Benutzer haben gemeinsamen Zugriff auf Cache.

Zwei Möglichkeiten

Fehlschlag im Cache: Beim ersten Mal oder nach Entfernung aus Cache muss der Prozess die Daten von der Platte lesen und in den Cache kopieren, bevor darauf zugegriffen werden kann.

Treffer im Cache: Die Daten werden direkt aus dem Cache gelesen.

Für Datenbankpuffer-Cache gibt es zwei Listen: LRU-Liste (Last Recently Used), MRU (Most Recently Used) und Dirty-Liste

Inhalt der LRU-Liste

freie Puffer:	Puffer, die nicht geändert wurden und daher verfügbar sind
fest verankerte Puffer:	Puffer, auf die momentan zugegriffen wird
dirty Puffer	geänderte Puffer, die noch nicht auf Platte geschrieben sind

Wenn ein Prozess einen Puffer findet oder darauf zugreift, wird der Puffer an den Anfang der LRU-Liste geschrieben (MRU)

Hinweis: Beim Seitenersetzungsverfahren LRU wird diejenige Seite verdrängt, auf die am längsten nicht mehr zugegriffen wurde. Intensiv genutzte Seiten werden mit hoher Wahrscheinlichkeit bei den nächsten Operationen wieder benötigt.

Exkurs: Seitenersetzungsverfahren

Realisierung als Keller-Speicher (Stack) oder mit Zähler.

Optimales Verfahren: Die Seite wird ersetzt, die in Zukunft am wenigsten gebraucht wird. Dieses Verfahren ist nicht realisierbar, ist jedoch das Maß?

Begriffe zu Festplattenzugriffszeit

- **rotational delay (Latenz-Zeit):** Zeit bis die gewünschte Seite unter dem Lesekopf auftaucht (im Durchschnitt ca. 4 msec),
- **seek time (Positionierung):** Zeit um den Lesekopf zu positionieren (2 msec bis 15 msec),
- **transfer time (Transferzeit):** Zeit um die Seite zu lesen (abhängig von der der Rotationszeit).

Beispiel:

- Speicherzugriffszeit (memory access): 100ns
- Festplatte: Latenz-Zeit: 8 ms 8.000.000 ns
- Festplatte: Positionierung: 15 ms 15.000.000 ns
- Festplatte: Transferzeit: 2 ms 2.000.000 ns

Die Zugriffszeit auf den Systempuffer ist ca. 100.000 mal schneller als die Zugriffszeit auf die Platte!

Daher: So wenig als möglich Festplattenzugriffe:

Einige Verfahren im Überblick - Basisverfahren):

First-in-First-out (FIFO)

- Referenz-String: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- Beispiel: 3 Speicherrahmen

1	1	4	5	
2	2	1	3	9 Seitenfehler
3	3	2	4	

- Beispiel: 4 Speicherrahmen

1	1	5	4	
2	2	1	5	10 Seitenfehler
3	3	2		
4	4	3		

Least Recently Used (LRU) Algorithmus

- Idee:
  - Extrapolation der Vergangenheit
  - Die am seltensten benutzte Seite wird ersetzt
- Beispiel:
  - Referenz-String: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- Implementierungen
  - Seiten-Referenz-Zähler
    - Je Seite: Zähler
    - Suche
  - Seiten-Stack
    - Seitenreferenz → Seite auf Stack
    - Unterste Seite: LRU
    - Keine Suche
  - Extrem aufwendig
    - In Praxis werden Approximationen verwendet

1		5	
2			8 Seitenfehler
3	5	4	
4	3		

Beispiel:

referenzierte Seiten / Seitenrahmen	/ 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 1 7 0	anzusprechende Seite
1	/ 7 7 7 2 2 2 2 4 4 4 0 0 0 1 1 1 1 1 1 1	einzulagende
2	/ / 0 0 0 0 0 0 0 0 3 3 3 3 3 0 0 0 0 0	bzw. vorhandene
3	/ / / 1 1 1 3 3 3 2 2 2 2 2 2 2 2 2 7 7	Seiten
Page-fault	/ ja ja ja ja / ja / ja ja ja ja / / ja / ja / / ja /	Seite war nicht vorhanden
ausgelagerte Seite	/ / / / 7 / 1 / 2 3 0 4 / / 0 / 3 / / 2 /	längst unbenutzte Seite

Anzahl der Seitenfehler: 12

Erläuterungen

Folgende Seitenreferenzierungen sind in drei Seitenrahmen unterzubringen: 70120304230321201170

Zu Beginn der ersten sieben Referenzierungen wird die Seite 0 drei mal angefordert, was dazu führt, dass diese in einem der Seitenrahmen erhalten bleibt. Die Seite 7 wird zu Beginn nur ein mal benötigt, was dazu führt, dass sie als erstes ausgelagert wird, da sie am häufigsten unbenutzt ist.

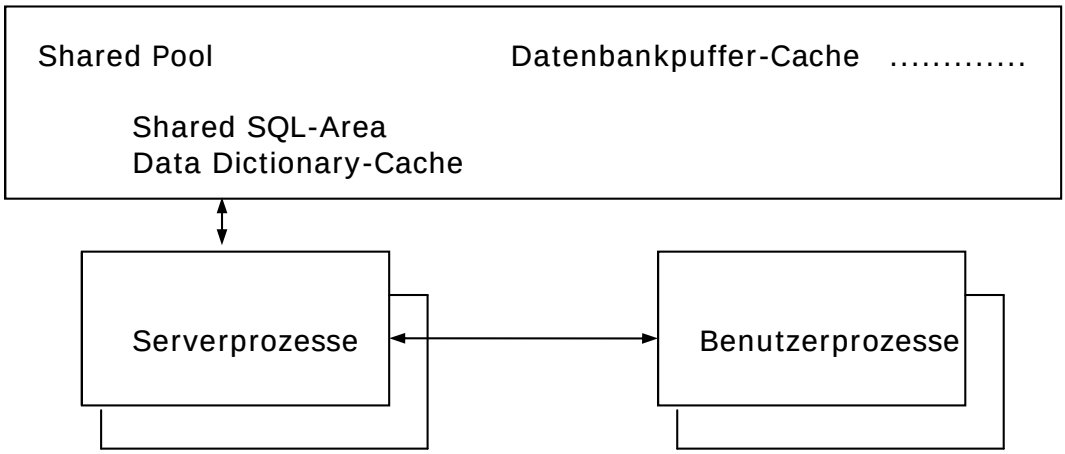
Ersetzungsstrategie

- Ersetzungskriterium: Zeit seit der letzten Referenzierung der Seite.
- Seiten im Systempuffer werden mit Hilfe eines Kellers verwaltet.
- Bei Referenz kommt die referenzierte Seite an die Kellerspitze.
- Falls die referenzierte Seite an Position n im Keller war, rutschen alle Seiten an den Positionen 1 bis n-1 eine Position tiefer.
- Seite am Kellerboden wird ersetzt.

Aufgabe:

Gegeben sei ein virtueller Speicher mit einem virtuellen Adressraum von 16 Seiten und einem physikalischen Adressraum von 4 Seiten, wobei letzterer leer sei. Ein gegebenes Programm referenziert virtuelle Seiten in folgender Reihenfolge: 0, 7, 2, 7, 5, 8, 9, 2, 4. Welche dieser Zugriffe verursachen einen Seitenfehler, wenn als Seitenersetzungsstrategie LRU bzw. FIFO benutzt wird?

2.3 Server- und Benutzerprozesse



Aufgaben der Serverprozesse: SQL-Anweisungen

- Parsen (Syntax, Sicherheit, Auflösung, Optimierung)
- Ausführen (physikalisches Lesen, ev. Ändern)
- Abrufen (Daten an Benutzerprozess weiterreichen)

Datenbankpuffer-Cache:

Fordert Benutzer Daten an, wird überprüft, ob Daten im Datenbankpuffer-Cache bereits vorhanden sind.

Datenblöcke:

Sind Daten noch nicht im Cache, liest der Server-Prozess die entsprechenden Blöcke aus der Datendatei in den Datenbankpuffer-Cache (DB_BLOCK_SIZE).

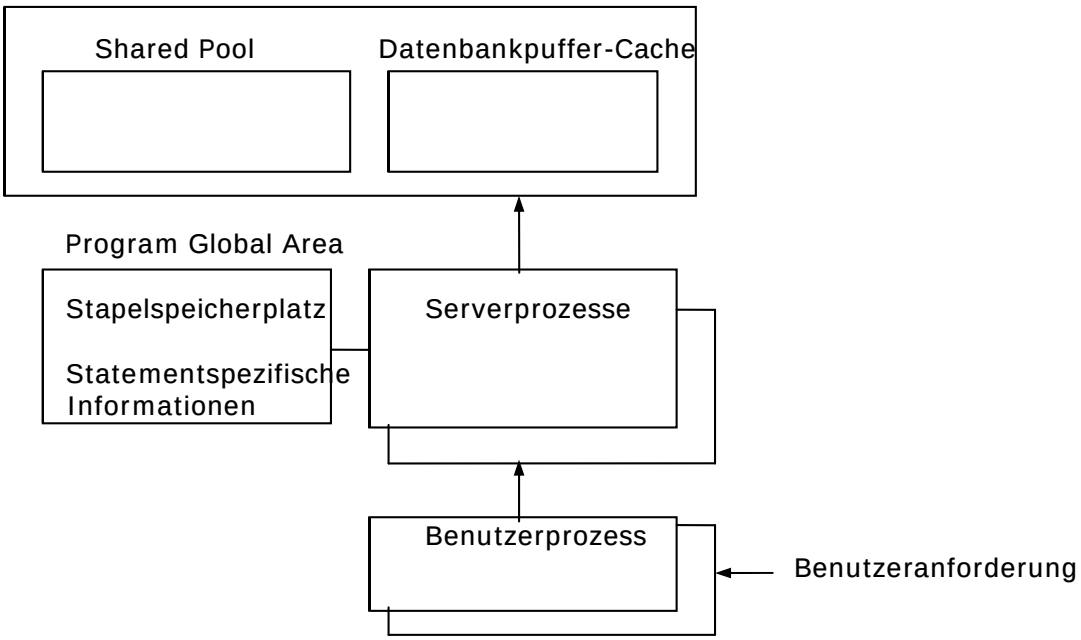
Werden Daten geändert, werden die Änderungen im Cache durchgeführt (After Image)

Rollback-Blöcke:

Die Originalversion wird in Rollback-Blöcken gespeichert (Before-Image). Vor Commit kann Transaktion zurückgerollt werden (Benutzer ändert Meinung)

2.4 Benutzeranforderungen (über SQL-Statement)

SGA



Server-Prozess

Wenn ein Server-Prozess einen Datenblock (ev. Seite) von der Platte in den Datenbankpuffer-Cache lesen muss,

- sucht er in der LRU-Liste
- sucht er einen freien Puffer
- stellt er dirty Buffer in die Dirty List

Er beendet die Suche, wenn er einen freien Puffer findet oder eine bestimmte Anzahl Puffer abgesucht hat, ohne einen freien Puffer zu finden.

Konfiguration der PGA

Speicherbereich, der Daten und Steuerinformationen für einen einzelnen Benutzer- oder Server-Prozess enthält. Die PGA wird vom ORACLE Server belegt, wenn ein Benutzer-Prozess sie belegt oder wenn ein Benutzer-Prozess sich bei einer Oracle-Datenbank anmeldet und eine Sitzung (Session) eingerichtet wird.

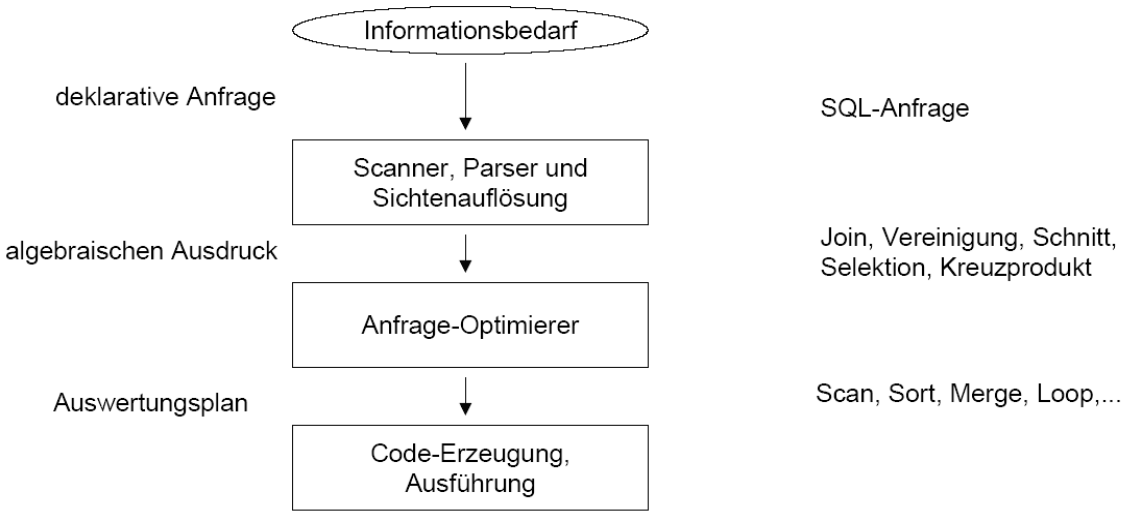
Eigenschaften der PGA

- Stapelspeicherplatz: für Sitzungsvariablen und Arrays
- Statementspezifische Informationen: für Benutzersitzung in Form von Private SQL-Area (PGA)
- Die PGA ist beschreibbar und non-shared

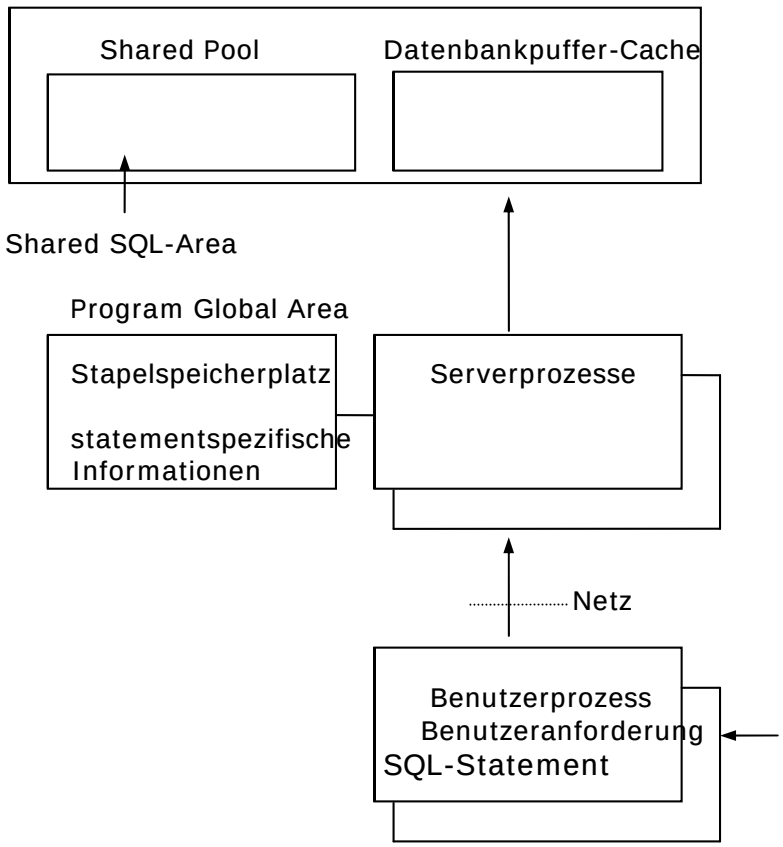
2.5 SQL-Anweisungen

2.5.1 Verarbeitung von SQL-Anweisungen

Übersicht:



SGA



2.5.2 Schritte bei SQL-Anweisungen

- **Parsen**
- **Execute**
- **Fetch**

ad 1) Parsen

- Syntax und Semantische Gültigkeit
- Data-Dictionary abfragen für Objekt-Auflösung, Privilegien und den besten Suchpfad (Parse-Tree)
- PGA für die Anweisung belegen

Hinweis: mehrere Serverprozesse können den Parse-Tree gemeinsam benutzen
Es wird ein Cursor verwendet (Speicherstruktur), um Statusinformationen für jede Anweisung aufzuzeichnen.

ad 2) Execute

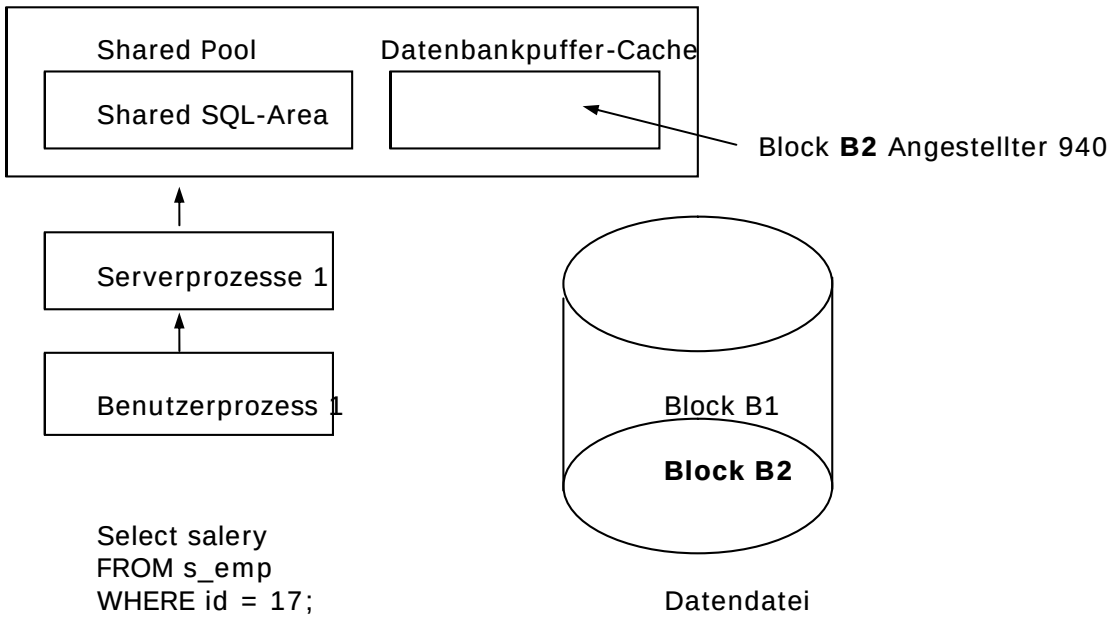
- Parse-Tree auf die Datenpuffer anwenden
- physikalisches oder logisches Lesen und Schreiben durchführen
- Constraint-Prüfung durchführen
- Daten, falls nötig, ändern

ad 3) Fetch

- Ergebniszeilen für eine SELECT-Anweisung liefern

2.5.3 SELECT-Anweisung

SGA



Select-Operation:

logisches Lesen, wenn Block im Speicher
physikalisches Lesen, wenn Block nicht im Speicher

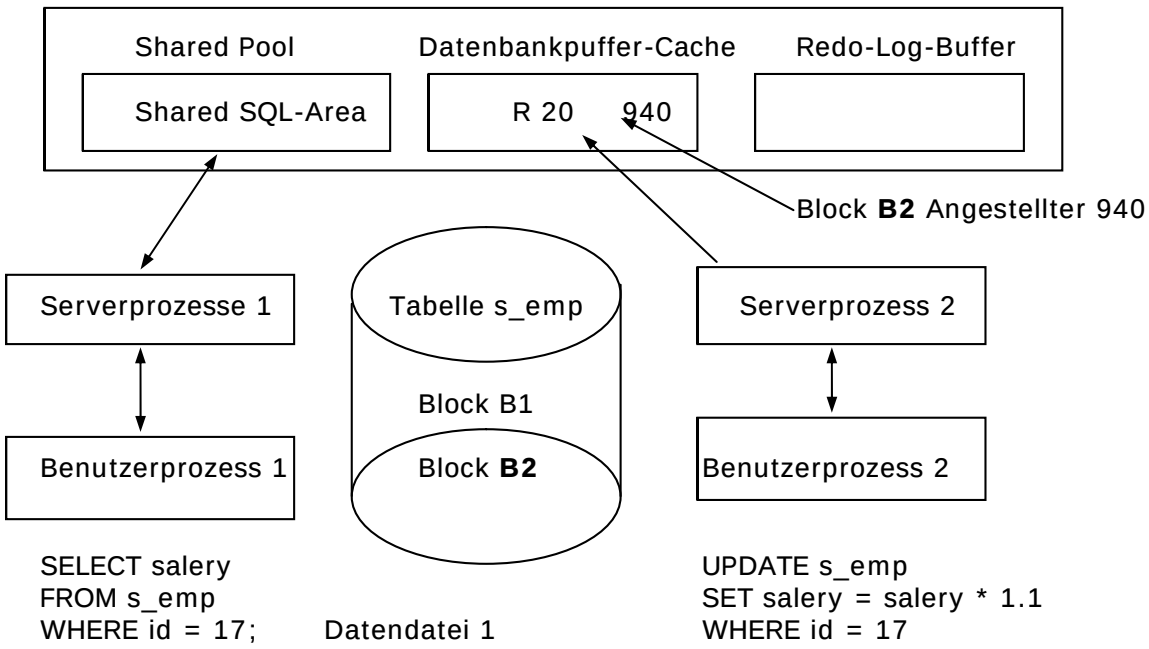
Blockgröße: 2k (Standard) bis 64 k (selten)

Der Oracle-Server liest bei einem Full-Table-Scan nach Möglichkeit mehrere Blöcke in den Speicher.

Hinweis: Zur Beantwortung einer Anfrage ist in der Regel ein so genannter *FULL TABLE SCAN* notwendig, d.h., die gesamte Tabelle muss Block für Block vom Plattenspeicher gelesen werden.

2.5.4 UPDATE-Anweisung

SGA



Dabei werden Rollback-Segmente belegt, um

- Lesekonsistenz
- Recovery
- Rollback

zu unterstützen.

Die UPDATE-Operation

- belegt Datenblöcke im Datenbankpuffer-Cache
- belegt Rollback-Blöcke im Datenbankpuffer-Cache
- setzt exklusive Zeilensperren auf Zeilen, die geändert werden sollen

- speichert einen Datensatz im Redo-Log-Puffer, um Before-Images und After-Images zu identifizieren
- sichert Rollback Daten in den Block-Puffer eines Rollback-Segments
- führt Änderungen im Datenblock-Puffer durch

Was wird gelesen, wenn Benutzer 1 die SELECT-Anweisung wiederholt?

Wenn Benutzer 1 eine SELECT-Operation nochmals ausführt, wird eine lesekonsistente Kopie der geänderten Zeile angelegt. Wenn ein Block dirty ist, erstellt der Serverprozess mit Hilfe des Rollback-Segments eine lesekonsistente Kopie der geänderten Zeile.

Block R 20 mit dem Wert 940 (salery) repräsentiert den Wert, den Benutzer 1 bei einer nachfolgenden Abfrage sieht (Rollback-Information).

Die Daten aller laufenden Transaktionen sind zusammen im Redo-Log-Puffer enthalten.
Die zwei Werte 949 und 1034 bilden einen atomaren Datensatz im Redo-Log-Puffer. Der Log-Datensatz enthält eine Speicheradresse für die Spalte, die Datei, Block, Zeile und Spalte beinhaltet.

Eigenschaften der Update-Operation

- Ein Rollback-Segment ist ein Objekt, das Rollback-Daten sichert.
- Der Oracle Server liefert lesekonsistente Kopie der Daten für alle Lesevorgänge.
- Eine Select Operation kann auch als Leseoperation bezeichnet werden. Lesevorgänge blockieren Schreibvorgänge nicht und umgekehrt. Erreicht wird dies durch die Rollback-Segmente (vgl. dazu da Multiversion Concurrency Konzept).

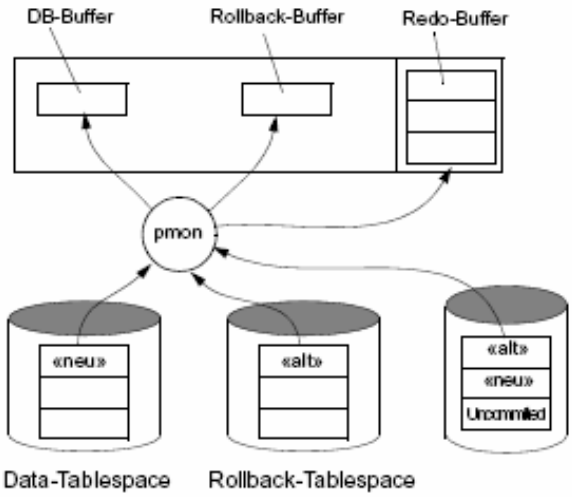
2.5.6 Ablauf eines Rollback

Durch das SQL Kommando ROLLBACK werden alle Transaktionen welche nicht commited sind rückgängig gemacht. Dazu werden die Rollbackbuffer aus den Rollbacksegmenten in der SGA wieder erstellt und der Rollback wird durchgeführt.

Beachte:

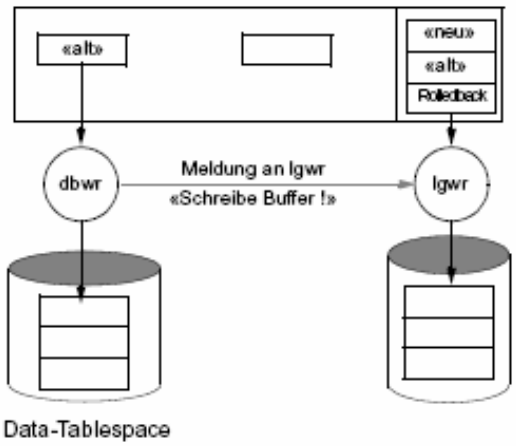
Das Redolog File gewährt in jedem Fall die Konsistenz, indem jeder Redo-Puffer gekennzeichnet ist mit «commited», «uncommited» oder «rolled back»

Laden der bereits geschriebenen Buffer in die SGA, Rollbackbuffer aus Rollbacksegmenten, Redobuffer aus Redologfile und Datenbuffer aus Tabelle.



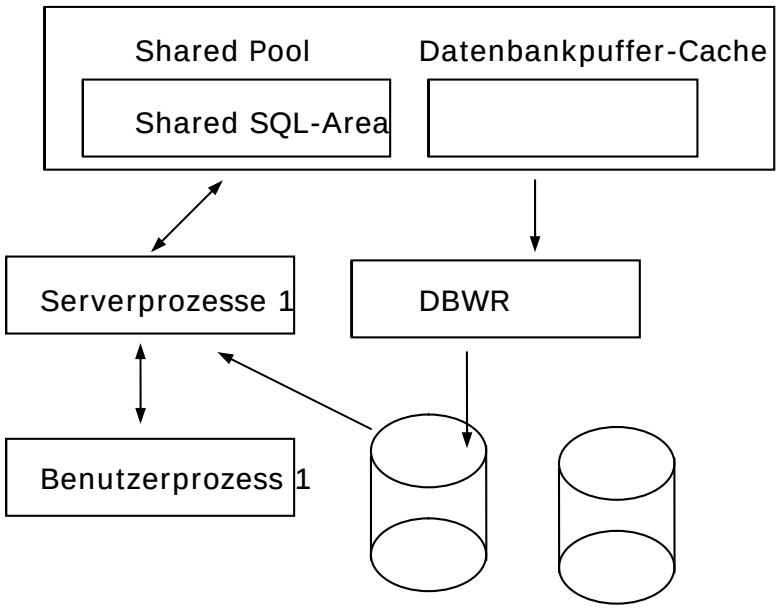
Rollback in der SGA durchführen, Redobuffer werden vertauscht und mit einem «rolled back» Record versehen. Der alte Wert im Rollbackbuffer wird in den Datenbuffer übertragen, er überschreibt den Wert des Datenbuffers.

Der *dbwr* Prozess schreibt den alten Wert in die Datenbank zurück, gleichzeitig schreibt auch der *lgwr* seinen Redobuffer ins aktive Redofile.



2.6 Der Data-Base-Writer-Prozess (DBWR)

SGA



Der DBWR verwaltet den Datenbankpuffer-Cache, damit Serverprozesse immer freie Puffer vorfinden.

DBWR-Prozess

- schriebt alle geänderten Puffer in Datendateien
- verwendet einen LRU-Algorithmus, um die häufig genutzten Blöcke im Speicher zu halten
- verzögert Schreibvorgänge zur E/A Optimierung

Der DBWR schreibt Dirty Puffer auf die Platte, wenn

- die dirty Liste eine bestimmte Länge erreicht
- ein Prozess eine bestimmte Anzahl von Puffern in der LRU- Liste abgesucht hat, ohne einen freien Puffer zu finden
- ein Time-Out auftritt
- eine DBWR-Checkpoint auftritt

2.7 Transaktionen

2.7.1 Einführung

Das Problem der Datenkontrolle (Data Control) betrifft 4 wesentliche Bereiche:

Recovery: Transaktionskonzept
Concurrency: Sperrkonzept
Security: Zugriffsrechte
Integrity: Constraints

2.7.2 Transaktionen bei Oracle

Hinweis: Zu Sperrstrategien vergleiche Kapitel 3. Hier erfolgt die Beschreibung des Ablaufs von Transaktionen auf der Serverseite.

Eine Folge von SQL-Anweisungen sind eine logische Einheit, daher eine Transaktion

Anfang einer Transaktion

- am Anfang einer Sitzung
- am Ende der vorhergehenden Transaktion

Ende einer Transaktion

Eine Transaktion wird **festgeschrieben** durch:

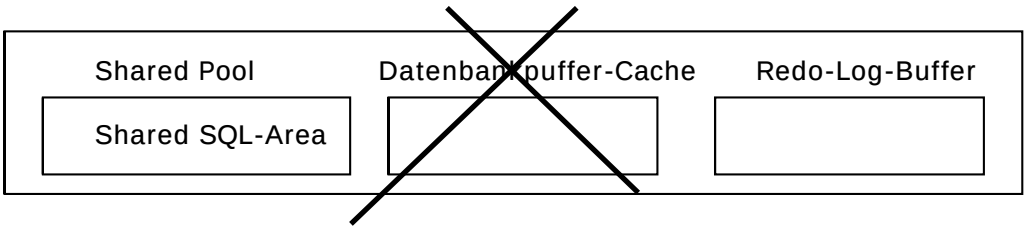
- COMMIT
- SQL-Anweisung die implizit ein Commit absetzt
- normales Ende des Benutzerprogramms mit Abmelden beim Oracle-Server

Eine Transaktion wird **abgebrochen** durch:

- ROLLBACK
- Abbruch auf Benutzeranfrage
- nicht normales Ende des Benutzerprogramms ohne Abmelden beim Server
- Prozessorfehler
- Plattenfehler

DDL und DCL-Anweisungen enthalten implizit ein Commit.

Transaktionen im Hauptspeicher



In einer Oracle DB werden Daten immer zuerst in den Hauptspeicher übertragen und geändert. Die Daten werden später zur permanenten Speicherung in Datendateien geschrieben.

Der DBWR schreibt periodisch die geänderten Blöcke in die Datendateien. Er schreibt häufig, aber nicht bei jedem COMMIT. Es kann daher sein, dass bereits festgeschriebene Daten noch im Datenbankpuffer-Cache stehen und die Gefahr besteht, dass bei einem Instancefehler alles im Hauptspeicher verloren geht.

Transaktionen protokollieren

Der Server zeichnet alle Änderungen der Datenbank im Redo Log-Puffer auf. Ein Hintergrundprozess, Log-Writer (LGWR) schreibt den Inhalt der Redo Log-Puffer auf Platte.

- Die Protokollierung dient Recovery-Zwecken.
- Der Redo-Log-Puffer ist ein zyklischer Puffer, der Informationen über Änderungen der Datenbank enthält.

Redo Log-Puffer

- Zeichnet die in der Datenbank durchgeführt Änderungen auf.
- Rekonstruiert Änderungen, die in der Datenbank und in Rollback-Segment-Einträgen durchgeführt wurden, wenn Recovery erforderlich ist.
- Kann mit dem Schlüsselwort UNRECOVERABLE in der Anweisung CREATE TABLE und CREATE INDEX umgangen werden.
- Kann mit dem ORACLE Loader umgangen werden.

Zur Sicherung der Daten im Datenbankpuffer-Cache gibt es Redo Log-Dateien. Sie zeichnen alle Änderungen von Daten im Datenbankpuffer-Cache auf.

Wenn ein Instancefehler auftritt, werden die geänderten Daten, die im Hauptspeicher waren, mit den Redo Log-Dateien wiederhergestellt. Die Redo Log-Dateien werden nur für Recovery verwendet.

2.7.3 Der Log Writer-Prozess (LGWR)

Der LGWR schreibt Redo Log-Einträge auf die Platte

Der LGWR schreibt in folgenden Fällen Redo Log-Puffer-Einträge in die Redo Log-Dateien:

- Commit
- Redo Log Puffer-Pool ist ein Drittel gefüllt
- LGWR Time-Out

Eigenschaften des LGWR-Prozesses

- Es gibt nur einen LGWR pro Instance.
- Ein Commit wird erst bestätigt, wenn die Transaktion in der Redo Log-Datei aufgezeichnet ist.
- Commits, die von anderen Benutzern durchgeführt werden, bevor der LGWR die Puffer wegen eines Benutzer-Commits wegschreibt, werden gesammelt (Huckepack Verfahren) Dadurch wird ein Durchschnitt von weniger als einer E/A pro Commit erreicht.
- Bei sehr langen Transaktionen kann es vorkommen, dass der Redo Log Puffer Pool mehr als ein Drittel gefüllt wird, bevor ihn der LGWR in die Redo Log-Datei schriebt.

2.7.4 Datenbank-Commits

Ablauf des Commit

- Ein Benutzer ruft COMMIT auf.
- Der LGWR schreibt den Redo Log-Puffer in die aktuelle Log-Gruppe.
- Der Benutzer wird benachrichtigt, dass die Transaktion festgeschrieben wurde
- Ressourcen-Sperren für Daten- und Rollback-Blöcke werden freigegeben.
- Der DBWR schreibt möglicherweise Datenbank-Blöcke auf Platte (incl. Rollback-Segment und Index-Blöcke.

Eigenschaften des Commit

- Der LGWR zeichnet Transaktionen laufend auf und ermöglicht damit dem DBWR, Schreibvorgänge zu verzögern. Dadurch wird E/A-Zeit für das physikalische Schreiben der geänderten Datenbank-Puffer-Blöcke aus dem Speicher gespart.
- Kurze Redo Log-Einträge enthalten geänderte Bytes und Informationen zur Identifizierung.
- Das Datenbank-Commit sammelt Redo Log-Einträge aus mehreren Transaktionen, die gleichzeitig ein Commit anfordern und schreibt sie in einen einzigen Schreibvorgang weg.
- Außer wenn der Redo Log-Puffer sehr gefüllt ist, wird maximal ein synchroner Schreibvorgang pro Transaktion benötigt.
- Die Größe der Transaktion beeinflusst die Dauer der eigentlichen Commit-Operation nicht.

2.7.5 Redo Log-Dateien

Redo Log-Dateien zeichnen alle Änderungen auf, die auf der Datenbank gemacht und werden für das Daten-Recovery verwendet. Wenn die Redo Log-Dateien vervielfältigt werden, wird die gleiche Redo Log-Information in mehrere Online Log-Dateien geschrieben.

Eigenschaften der Redo Log-Dateien

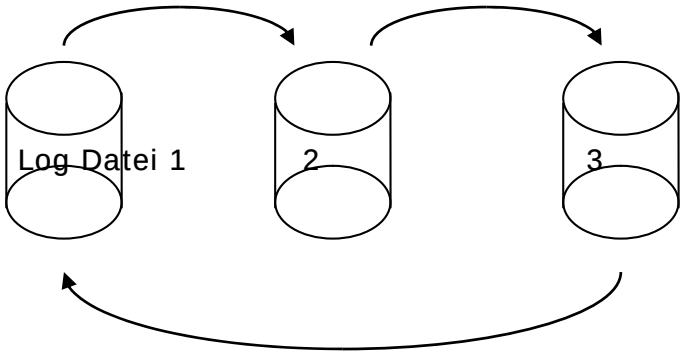
- Redo Log-Dateien werden zyklisch beschrieben.
- Es müssen mindestens zwei Redo Log-Gruppen existieren.
- Redo Log-Dateien sollten auf das schnellste und zuletzt geladene E/A-Gerät geschrieben werden.

Multiplex Redo-Log-Dateien

Die empfohlene Konfiguration für Redo Log-Dateien beinhaltet mindestens zwei Redo Log-Member pro Gruppe, wobei sich jedes Member auf einer anderen Platte befindet.

- Multiplex Redo-Log-Dateien schützen vor einem Verlust der Redo-Log-Dateien.

- Alle Member einer Gruppe von Log-Dateien enthalten die gleiche Information und besitzen die gleiche Größe.
- Member einer Gruppe werden gleichzeitig aktualisiert.
- Jede Gruppe sollte die gleiche Anzahl Member enthalten.



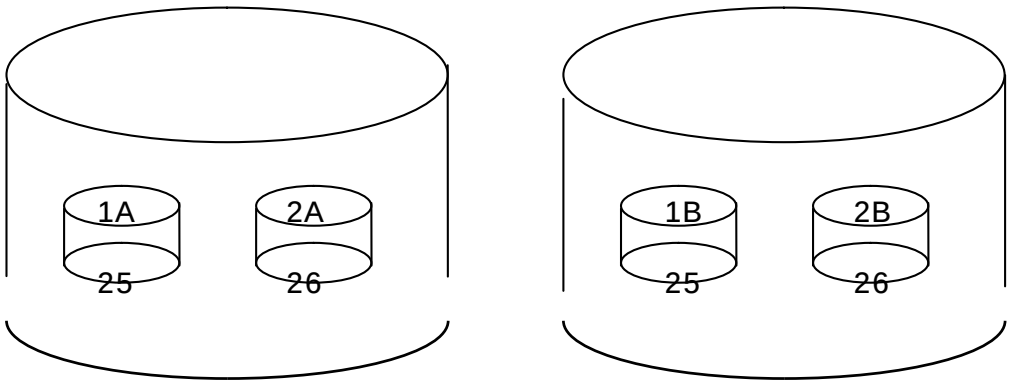
Beispiel: Multiplex-Redo-Log-Dateien (Oracle 8)

Platte 1 und 2 stellen unterschiedliche Platten auf der gleichen Maschine dar. Jedes Member einer Gruppe muss die gleiche Größe haben. Jedes Member einer Gruppe sollte auf verschiedenen Platten gespeichert sein, um gegen Ausfall geschützt zu sein. Alle Member einer Gruppe werden gleichzeitig beschrieben.

Die Standardinstallation erzeugt zwei Redo Log-Gruppen mit jeweils einem Member pro Gruppe und mit einer Standardgröße von 500 KB. Weitere Member auf verschiedenen Platten können hinzugefügt werden. Die Vergrößerung der Redo Log-Dateien ist ebenfalls empfehlenswert.

Hinweis: Mehr als zwei oder drei Dateien anlegen macht keinen Sinn.

Log-Switch



Redo Log-Gruppe 1
Member 1A, Platte 1
Member 1B, Platte 2
Log Sequence 25

Redo Log-Gruppe 2
Member 2A, Platte 1
Member 2B, Platte 2
Log Sequence 25

- Ein Log-Switch tritt auf, wenn der LGWR das Schreiben in eine Redo Log-Gruppe beendet und anfängt, in eine andere zu schreiben.
- Ein Log-Switch tritt auf, wenn der LGWR eine Log-Datei-Gruppe gefüllt hat.
- Bei einem Log-Switch wird der aktuellen Redo Log-Gruppe eine Sequence-Nummer zugewiesen, die die Informationen in dieser Redo Log-Gruppe identifiziert und zur Synchronisierung verwendet wird.
- Ein Log-Switch kann vom DBA mit dem Befehl ALTER SYSTEM SWITCH LOGFILE erzwungen werden.
- Bei einem Log-Switch wird automatisch ein Checkpoint gesetzt.
- Die Arbeit kann weitergeführt werden, solange noch mindestens ein Member einer Gruppe verfügbar ist. Wenn ein Member beschädigt oder nicht verfügbar ist, werden Meldungen in die Trace-Datei des LGWR und in die Alert-Datei geschrieben.

Sequence-Nummer

- Jede Redo Log-Datei wird eindeutig durch ihre Sequence-Nummer identifiziert.
- Beim Recovery verwendet Oracle die Redo Log-Dateien in aufsteigender Reihenfolge entsprechend ihrer Sequence-Nummer,
- Die aktuelle Log Sequence-Nummer wird in der Kontrolldatei gespeichert.

2.8 Checkpoints

Während eines Checkpoints schreibt der DBWR alle dirty Puffer des Datenbankpuffer- Cache auf Platte. Dadurch ist sichergestellt, dass alle Datenblöcke, die seit dem vorhergehenden Checkpoint geändert wurden, zu diesem Zeitpunkt auf Platte geschrieben sind.

Wann treten Checkpoints auf?

- Bei jedem Log-Switch (kann nicht unterdrückt werden).
- Eine festgelegte Anzahl von Sekunden nach dem letzten Datenbank-Checkpoint (LOG_CHECKPOINT_TIMEOUT).
- Wenn eine festgelegte Anzahl von Redo Log-Blöcken seit dem letzten Checkpoint auf Platte geschrieben wurde (LOG_CHECKPOINT_INTERVAL).
- Beim Herunterfahren der Instance, sofern die Instance nicht abgebrochen wird.
- Wenn der DBA einen Checkpoint erzwingt (ALTER SYSTEM CHECKPOINT).
- Wenn ein Tablespace offline gesetzt wird und mindestens eine seiner Dateien vorher online war.

Synchronisierung

- Bei jedem Checkpoint wird die Sequence-Nummer des Checkpoint in allen Datenbank-Datei-Headern und in der Kontrolldatei aktualisiert.
- Die Kontrolldatei bestätigt während des Hochfahrens der Instance, dass alle Dateien dieselbe Checkpoint-Nummer haben.
- Der Parameter LOG_CHECKPOINT_TIMEOUT gibt die Zeitspanne an, nach der ein neuer Checkpoint auftritt.

– Der Parameter LOG_CHECKPOINT_INTERVAL gibt an, wie viele neu gefüllte Blöcke der Redo Log-Datei einen Checkpoint auslösen.

Kontrolldatei (Z. B. Name: initorcl.ora)

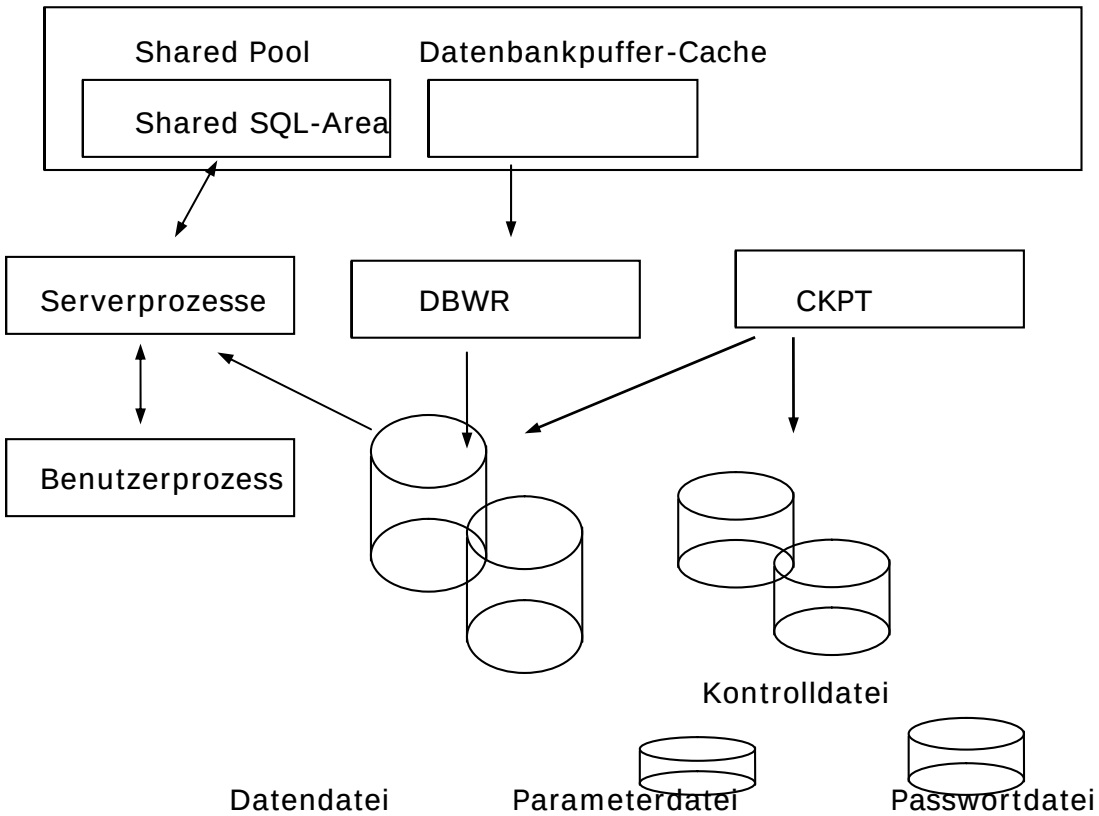
Die Kontrolldatei ist eine kleine binäre Datei, die die Struktur der Datenbank beschreibt

Kontrolldatei-Eigenschaften:

– Alle notwendigen Datenbankdateien und Log-Dateien sind in der Kontrolldatei aufgeführt.
– Der Name der Datenbank wird in der Kontrolldatei gespeichert.
– Die Kontrolldatei wird benötigt um die Datenbank zu mounten, zu öffnen und auf sie zuzugreifen.
– Die Kontrolldatei speichert auch Synchronisierungsinformation, die für Recovery benötigt wird.
– Die empfohlene Konfiguration besteht aus mindestens zwei Kontrolldateien auf verschiedenen Platten.
– Der Parameter CONTROL_FILES identifiziert die Kontrolldatei (Parameterdatei).
– Die Kontrolldatei muss für Schreibzugriffe des Oracle Servers verfügbar sein, wenn die Datenbank geöffnet ist.
– Der Standardname der Kontrolldatei ist betriebssystemabhängig.

Checkpoint-Prozess (CKPT)

SGA



Datenbank-Checkpoints stellen sicher, dass alle geänderten Datenbankpuffer in die Datenbankdateien geschrieben sind. Die Datenbankdateien werden mit Zeitstempel als aktuell markiert. Der Checkpoint wird in der Kontrolldatei aufgezeichnet.

Checkpoints

Der DBA kann mit dem Befehl ALTER SYSTEM CHECKPOINT einen Checkpoint erzwingen.

CKPT-Prozess

- Der Checkpoint-Prozess wird durch den Parameter CHECKPOINT_PROCESS = TRUE aktiviert. Der Default-Wert ist FALSE.
- Wenn dieser Hintergrundprozess aktiviert ist. übernimmt er vom LGWR die Aufgabe, Dateien beim Checkpoint zu aktualisieren.
- Die Header der Daten- und Kontrolldateien werden am Schluss des Checkpoints aktualisiert.
- Häufigere Checkpoints verringern die Zeit, die für das Recovery nach einem Instance-Fehler benötigt wird. Können aber zu Performance-Einbußen führen.
- Der Checkpoint-Prozess kann die Datenbank-Performance verbessern, wenn die Datenbank viele Datenbankdateien hat.

2.9 Die Hintergrundprozesse PMON, SMON und ARCH

Es gibt unterschiedliche Gründe für eine Unterbrechung des normalen Datenbankbetriebes:

PMON

Ein Benutzerprozess kann irregulär beendet werden (CRTL/C). In diesem Fall kann der Datenbank-Betrieb dadurch behindert werden, dass der nicht mehr vorhandene Benutzerprozess Ressourcen blockiert (z.B. gesperrte Datensätze) und evtl. Transaktionen nicht beendet sind. Durch ein Ereignis dieser Art ist der Datenbankbetrieb nicht gefährdet es muss allerdings dafür gesorgt werden, dass die Blockierung aufgehoben wird.

Der Prozess-Monitor (PMON) stellt irreguläre Beendigungen von Prozessen fest und kann blockierte Ressourcen wieder freigeben. Zudem rollt er Transaktionen zurück. Die durch die abgebrochenen Benutzerprozesse begonnen wurden aber noch nicht mit Commit bestätigt waren.

- Bereinigt nicht normal beendete Verbindungen.
- Rollt nicht fest geschriebene Transaktionen zurück.
- Gibt Sperren frei, die von einem beendeten Prozess gehalten werden.
- Gibt SGA-Ressourcen frei, die von einem abgebrochenen Prozess belegt sind.

SMON

Hintergrundprozesse werden irregulär beendet (Stromausfall). Dieses Ereignis zerstört die Instance und kann einen inkonsistenten Zustand der Datenbank zur Folge haben. Es muss also dafür gesorgt werden, dass die verloren gegangenen Daten rekonstruiert werden.

Der System Monitor (SMON) überwacht das Datenbanksystem und wird nach einem Neustart der Instance aktiv. Befindet sich die Datenbank in einem nicht konsistenten Zustand, führt er auf Grund der Einträge in den Online Redo Log-Dateien ein Instance Recovery durch.

- Führt automatisches Instance Recovery durch.
- Gibt Speicherplatz frei, der von nicht länger benötigten temporären Segmenten beleg wird.
- Verbindet zusammenhängende freie Speicherbereiche in den Datendateien.

Archiver-Prozess

Externe Speichermedien werden zerstört, so dass größere Datenmengen verloren gehen. In diesem Falle können die Daten nicht allein auf Grund der Einträge in den Online Redo Log-Dateien wiederhergestellt werden. Die Redo Log Information reicht zeitlich nicht weit genug zurück. Es ist eine zusätzliche Sicherung in Form eines Backups der Datendateien zur Verfügung zu stellen. Dabei tritt allerdings das Problem auf, dass das Backup meist nicht auf dem aktuellen Stand ist, Die bisher durchgeführten Änderungen der Daten lassen sich vielmehr in drei Gruppen aufteilen.

- Veränderungen, die nach dem Anlegen der Datenbank und vor dem Erstellen des Backups durchgeführt wurden, sind im Backup enthalten.
- Veränderungen, die nach dem Backup und vor dem durch die Redo Log-Dateien abgedeckten Zeitraum durchgeführt wurden, sind verloren. Sie sind weder im Backup noch in den Redo Log-Dateien enthalten.
- Veränderungen, die innerhalb des durch die Redo Log-Dateien abgedeckten Zeitraumes durchgeführt wurden, sind in den Redo Log-Dateien enthalten und können nachgefahren werden.
- Zwischen dem Backup und dem durch die Redo Log-Dateien abgedeckten Zeitabschnitt entsteht also eine Lücke. Es kann somit nicht auf den aktuellen Stand nachgefahren werden.

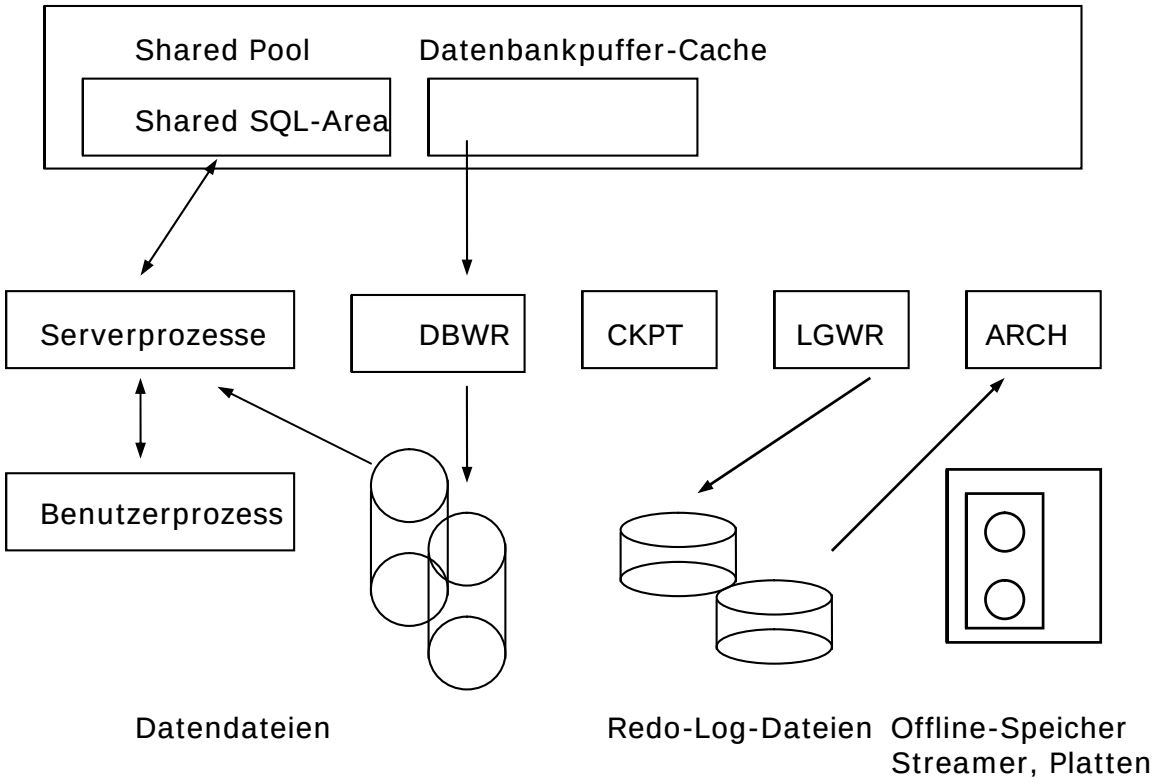
Der ARCH-Prozess (optional) sorgt dafür, dass Online Redo Log-Dateien, bevor sie wieder überschrieben werden, physisch kopiert und damit gesichert sind. Dadurch entsteht für jede Online Redo Log-Datei eine physische Sicherung in Form einer 0nline Redo Log-Datei.

Die Offline Redo Log-Dateien sind nach einem Plattencrash und daraus resultierendem Media Recovery in der Lage, die Lücke zwischen den durch das Backup und die Online Redo Log-Dateien abgedeckten Bereichen zu füllen. Dadurch lässt sich die Datenbank wieder in den aktuellen, konsistenten Zustand bringen.

Aktueller Stand der DB (vgl. Kapitel Backup&Recovery) ergibt sich aus:

Cold Backup + Offline-Redo-Log-Dateien + Online-Redo-Log-Dateien

SGA



Der Archiver-Prozess ARCH kopiert Online Redo Log-Dateien auf ein festgelegtes Speichergerät, wenn der LGWR auf eine neue Gruppe wechselt (Log-Switch).

Der Archiver-Prozess

- kopiert für Media Recovery Redo Log-Dateien auf Band oder Platte
- arbeitet nur, wenn ein Log-Switch auftritt
- ist optional und wird nur im ARCHIVELOG-Modus benötigt
- kann auf Band oder Platte schreiben
- Die Grafik oben verdeutlicht die Strukturen und Prozesse der Transaktions-Protokollierung und zeigt, wie sie zusammenwirken.

2.10 Die Prozesse RECO, LCKn, Pnnn und SNPn

Die Prozesse RECO, LCKn, Pnnn und SNPn werden nur dann erzeugt, wenn eine der Zusatz-Optionen verwendet wird.

Prozesse:

- RECO behebt Fehler, die eine verteilte Transaktion betreffen,

- LCKn führt Sperren zwischen Instances in einem Parallel Server System durch.
- Pnnn unterstützt parallele Abfragen (Parallel Query), parallele Erstellung eines Index,
- Paralleles Laden von Daten und paralleles CREATE TABLE AS SELECT.
- SNPn aktualisiert Snapshots (schreibgeschützte Replikations-Tabellen) automatisch. Er ist ebenfalls für die Warteschlangen der Serverjobs und der Replikationen zuständig.

Optionen:

- Die Parallel Query-Option ist ein getrennt lizenziertes Produkt.
- Die Procedural-Option ist für Snapshots erforderlich. Sie ist im Oracle8/Server enthalten.
- Die Distributed-Option wird für verteilte Transaktionen benötigt. Sie ist ein getrennt lizenziertes Produkt.
- Die Replication-Option ist ein getrennt lizenziertes Produkt.

2.11 Parameterdatei

Die Parameterdateien definieren die Instance. Oracle muss eine Parameterdatei init<SID>.ora lesen, um eine Instance zu starten. Eine Parameterdatei ist eine Textdatei, die eine Liste der Konfigurationsparameter der Instance enthält. die belegen diese Parameter mit bestimmten Werten. Um die Speichergröße und Prozesseinstellungen einer Oracle Instance festzulegen.

Neben anderen Einstellungen, legt die Parameterdatei folgendes fest: Wie viel Speicher für die Speicherstrukturen der SGA verwendet wird SGA mit gefüllten Online Redo Log-Dateien geschieht

SGA-Größe

Parameter	Beschreibung
SHARED_POOL_SIZE	Größe (in Byte) des Bereichs, der für Shared SQL und für PL/SQL-Anweisungen bestimmt ist
DB_BLOCK_SIZE	Größe (in Byte) eines einzelnen Datenblocks und Datenbankpuffers
DB_BLOCK_BUFFERS	Anzahl der Datenbankpuffer, mit jeweils der Größe DB_BLOCK_SIZE, die für die SGA belegt werden. (Der Gesamtbetrag an Speicher, der für den Datenbankpuffer-Cache in der SGA belegt wird, ergibt sich aus DB_BLOCK_SIZE mal DB_BLOCK_BUFFERS.)
LOG_BUFFER	Anzahl Bytes, die für den Redo Log-Puffer belegt werden

2.12 Trace-Dateien und Alert-Dateien

Tritt während des Ablaufs einer Oracle Instance ein Fehler auf, werden Meldungen in die **Alert**-Datei geschrieben.

Wird der Fehler von einem Server- oder Hintergrundprozess entdeckt, wird die Information in eine **Trance**-Datei geschrieben.

Alert-Datei

Die Alert-Datei einer Datenbank ist ein chronologisches Protokoll der Meldungen und Fehler. Die Alert-Datei enthält folgende Informationen:

- alle internen Fehler (ORA-600)), Fehler: korrupter Block (ORA-1578) und Deadlock-Fehler (ORA-60)
- Verwaltungs-Operationen (DDL) und Server Manager-Anweisungen (STARTUP, SHUTDOWN, ARCHIVE LOG und RECOVER)
- die Werte aller nicht standardmäßigen Initialisierungsparameter zum Zeitpunkt des Starts der Datenbank und der Instance
- Oracle verwendet die Alert-Datei als Alternative, um solche Informationen an der Operator-Konsole auszugeben. Der Speicherplatz für die Alert-Datei wird mit dem Parameter BACKGROUND_DUMP_DEST festgelegt

Es ist wichtig, die Alert-Datei täglich zu überprüfen, um Probleme rechtzeitig zu entdecken, bevor sie ernster werden und solange noch ein gültiges Backup existiert.

Trace-Dateien

Trace-Dateien entdeckt ein Server- oder Hintergrundprozess einen internen Fehler, wird die Information in die zugehörigen Trace-Datei geschrieben. Der Ort für die Trace-Dateien steht im Parameter BACKGROUND_DUMP_DEST, wenn die Information von einem Hintergrundprozess geschrieben wird und im Parameter USER_DUMP_DEST, wenn die Information von einem Server-Prozess geschrieben wird.

3 Sperrmodell

3.1 Lesekonsistenz

Während der gesamten Verarbeitung einer Anweisung liefert der Oracle Server einen Snapshot der Tabellendaten, der zu einem am Anfang der Anweisung gemachten Zeitstempel konsistent ist. Lesekonsistenz bedeutet, dass Daten, die ein Benutzer liest oder ändert, konsistent bleiben, bis der Benutzer damit fertig ist. Nicht festgeschriebene Änderungen darf nur der Benutzer sehen, der die Änderungen gemacht hat. Ein Benutzer, der eine Abfrage durchführt, darf nur die Daten sehen, die zu Anfang der Abfrage (Default-Einstellung oder zu Anfang der Transaktion (Befehl SET TRANSACTION READ ONLY) festgeschrieben waren.

Anmerkung: Snapshot ist eine lesekonsistente Abfrage zum Beginn der Abfrage und wird ev. durch die Verwendung der alten Werte ermöglicht.

Beispiele für Lesekonsistenz

Die Sekretärin der Abteilung EDVO will die Anzahl der Einheiten für jeden Gegenstand dieser Abteilung wissen.

```
SQL> select Gegenstands_ID, Title, Einheiten
2 from Gegenstand
3 where
4 Abteilung = 'EDVO'
5 order by Gegenstands_ID;

Gegenstands_ID TITLE Einheiten
---------- ------------------------------- -----
101 Mathematik I 3
177 Geografie I 3
178 ADAT 3 3
```

Direkt nach dem die Abfrage der Sekretärin an die Datenbank abgeschickt wurde, änderte die Sekretärin im Hauptgebäude alle Gegenstände mit drei Einheiten auf vier. Konsistenz auf der **Ebene der Anweisungen** bedeutet, dass die Abfrage keine Zeile zurückgeben kann, in der die Zahl der Einheiten auf vier aktualisiert wurde. Je nachdem, wann die Sekretärin die Änderungen festschreibt, sieht sie die Sekretärin der EDVO oder er sieht sie nicht.
Sie wird jedoch in einer einzelnen SQL-Anweisung niemals eine teilweise durchgeführte Änderung sehen.

Das Lesekonsistenz-Beispiel verwendet folgende synchronisierte Unterabfrage:

```
SQL>SELECT first_name, dept_id, salary
2       FROM s_emp a
3       WHERE salery > (SELECT AVG(salary) FROM s_emp
4       WHERE b.dept_id = a.dept_id);
```

Der Server garantiert, dass die obige Anweisung das gleiche Ergebnis hat, auch wenn ein anderer Benutzer gleichzeitig Änderungen auf S_EMP festschreibt.

Der Oracle Server liefert nur festgeschriebene Daten oder Änderungen, die der aktuelle Benutzer zuvor durchgeführt hat; siehe folgendes Beispiel:

Zeit	Benutzer A	Benutzer B
0	UPDATE s_emp	
1		SELECT ... FROM s_emp b
2	COMMIT	
3		SELECT ... FROM s_emp a

Obwohl Oracle Konsistenz innerhalb einer einzelnen SQL-Anweisung bietet, garantiert sein **Standardverhalten** keine Lesekonsistenz über mehr als eine Anweisung. Wird eine Tabelle zweimal abgefragt, kann es sein, dass man beim zweiten Mal andere Resultate erhält, wenn ein anderer Oracle-Benutzer die Tabelle zwischen der ersten und der zweiten Abfrage ändert.

Es können Situation entstehen, in denen Lesekonsistenz für mehr als eine Anweisung benötigt wird. Es kann sein, dass Lesekonsistenz für eine bestimmte Transaktion gebraucht wird.

Sperrreihenfolge - Übersicht

Der Oracle Server erlaubt gleichzeitigen Zugriff auf Daten indem er die Datenintegrität durch Sperren auf Daten und Data Dictionary gewährleistet.

- Lesevorgänge blockieren Schreibvorgänge nicht
- Schreibvorgänge blockieren Lesevorgänge nicht.
- Lesekonsistenz wird durch Rollback -Segment-Einträge gewährleistet
- Sperren auf Zeilenebene (Default) und auf Tabellenebene werden unterstützt.
- Sperren können automatisch und manuell gesetzt werden.
- Shared oder Exclusive Locks können manuell gesetzt werden.
- Sperren auf Zeilenebene sind für jede Zeile der Datenbank verfügbar.
- Der Oracle Server weitet Sperren nicht aus, z.B. von Zeilen- auf Tabellenebene.

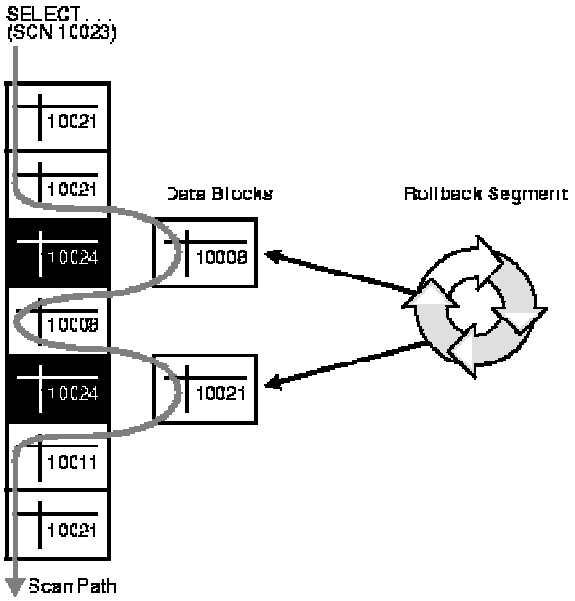
3.2 Multiversion Concurrency Control System

Oracle verwaltet die Lesekonsistenz innerhalb einer Abfrage automatisch. Es ist damit sichergestellt, dass alle Daten, die eine Abfrage sieht, von genau einem einzigen Zeitpunkt stammen (Statement-level read consistency).

Oracle kann zusätzlich die Lesekonsistenz aller Abfragen innerhalb einer Transaktion sicherstellen (Transaction-level read consistency).

Zur Sicherstellung der Lesekonsistenz verwendet Oracle Rollback Segmente. Ein Rollback Segment enthält die alten Werte von jenen Daten, die von Transaktionen verändert und erst kürzlich oder noch gar nicht kommittiert wurden.

Die Abbildung zeigt, wie Oracle Lesekonsistenz auf Aufrufebene unter der Verwendung des Rollback Segments erreicht.



Erklärung der Grafik:

Sobald eine Abfrage die Ausführungsstufe erreicht, wird die aktuelle System Change Number (SCN) bestimmt. In der oben angeführten Abbildung ist diese Zahl 10023. Wird nun durch die Abfrage auf Blöcke lesend zugegriffen, so werden nur jene Blöcke benutzt, welche niedrigere SCN s haben, als die SCN der Abfrage.

Blöcke, die geänderte Daten enthalten (jüngere SCN s) werden aus den Rollback Segmenten rekonstruiert und diese Daten dann an die Abfrage zurückgegeben. Hierzu gibt jede Abfrage alle kommittierten Daten hinsichtlich der bei Beginn der Ausführung festgelegten SCN Nummer zurück.

Änderungen anderer Transaktionen, die während der Durchführung einer Abfrage durchgeführt werden, werden nicht berücksichtigt. So wird garantiert, dass jede Abfrage konsistente Daten liefert.

3.3 Die Auswahl eines Isolationlevels

Bei Oracle stehen drei Isolation Level zur Verfügung (Automatisches Setzen von Sperren):

READ COMITTED	(Standard, kann nicht unterschritten werden auf READ UNCOMITTED) Dies bedeutet Statement Level Read Consistency
SERIALIZABLE	Dies bedeutet Transaction Level Read Consistency
READ ONLY	Dies ist für DATA-Warehouse Applications z.t. sinnvoll.

Hinweis: SQL92 definiert vier Isolation Levels:

Isolation Level	Dirty Read	NonRepeatable Read	Phantom Read
Read uncommitted	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible
Repeatable read	Not possible	Not possible	Possible
Serializable	Not possible	Not possible	Not possible

Hinweis: SQL-2 schließt das Auftreten eines Lost Update von vornherein aus!

Allgemein gilt: Je höher der Isolationsgrad, desto geringer die Performance und umgekehrt.

Einstellung des Isolation-Levels?

SET TRANSACTION ISOLATION LEVEL

SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET TRANSACTION READ ONLY;

Um den Aufwand für Netzwerk und Verarbeitung zu verringern, kann man anstatt bei jeder Transaktion den Isolationlevel zu setzten durch das Kommando ALTER SESSION diesen sofort für alle nachfolgenden Transaktionen automatisch setzen.

ALTER SESSION SET ISOLATION_LEVEL SERIALIZABLE;
ALTER SESSION SET ISOLATION_LEVEL READ COMMITTED;

Die Darstellung fasst die grundlegenden Unterschiede zwischen 'read committed' und 'serializable' - Transaktionen zusammen:

	Read Committed	Serializable
Dirty write	Not possible	Not possible
Dirty read	Not possible	Not possible
Nonrepeatable read	Possible	Not possible
Phantoms	Possible	Not possible
Compliant with ANSI/ISO SQL 92	Yes	Yes
Read snapshot time	Statement	Transaction
Transaction set consistency	Statement level	Transaction level
Row-level locking	Yes	Yes
Readers block writers	No	No
Writers block readers	No	No
Different-row writers block writers	No	No
Same-row writers block writers	Yes	Yes
Waits for blocking transaction	Yes	Yes

Subject to "cannot serialize access"	No	Yes
Error after blocking transaction aborts	No	No
Error after blocking transaction commits	No	Yes

Anwendungsentwickler und Designer sollten den Isolationlevel unter Beachtung der Anwendungsperformance, der gewünschten Konsistenz und den Programmieranforderungen wählen.

Für Umgebungen, in denen viele gleichzeitig arbeitende User Transaktionen absetzen, müssen die Performanceanforderungen der Transaktion unter Beachtung der zu erwartenden Transaktionen und die Forderungen an das Antwortzeitverhalten abgeschätzt werden.

Alle Oracle Isolation Modes liefern durch die Verwendung von Row-Level-Locks und des Multiversion Concurrency Control Systems einen hohen Grad an Konsistenz, Parallelität, Gleichzeitigkeit und Performance.

Leser und Schreiber sperren sich nicht gegenseitig. Daher ist es möglich, dass Abfragen konsistente Daten sehen und trotzdem ein hoher Grad an Parallelität geboten werden kann.

Die Wahl des Isolation Levels SERIALIZABLE

Der SERIALIZABLE Isolationlevel ist vorwiegend für Umgebungen geeignet, in denen es relativ unwahrscheinlich ist, dass zwei Transaktionen gleichzeitig dieselbe Zeile modifizieren wollen und lang dauernde Transaktionen vorwiegend READ ONLY sind. Am besten eignet sich dieser Isolationlevel daher für Umgebungen mit großen Datenbanken und kurzen Transaktionen, die nur wenige Zeilen verändern.

Im Gegensatz zu anderen Implementierungen von Datenbanken die Blöcke sowohl beim Lesen als auch beim Schreiben bei Verwendung des Isolation Level SERIALIZABLE sperren, bietet Oracle so genannte **Nonblocking Queries** und Sperren auf Zeilenebene. Beide helfen bei der Reduzierung der Konflikte bei zwei schreibenden Transaktionen.

Bei Anwendungen, bei denen es oft zu Konflikten zwischen lesenden und schreibenden Transaktionen kommt, erlaubt der SERIALIZABLE Isolationlevel erheblich mehr Durchsatz als bei anderen Systemen (Portierbarkeitsproblem, vgl. Multiversion Concurrency Control System).

Anwendungen, die Zusammenfassungen von Daten generieren und in der Datenbank abspeichern, sollten den SERIALIZABLE Modus verwenden, da dieser dieselbe Konsistenz bietet, die man auch beim READ ONLY Modus hat und außerdem INSERT, UPDATE und DELETE Anweisungen erlaubt.

Transaktionen, die DML mit Unterabfragen verwenden, sollten SERIALIZABLE verwenden um Konsistenz beim Lesen zu garantieren.

SERIALIZABLE Transaktionen verlangen zusätzliche Programmiertätigkeit (abfangen/abfragen von Cannot serialize access Error und veranlassen von Rollback und Retry der Transaktion). Wie auch bei anderen Datenbanksystemen ist es nötig extra Code zu schreiben um Deadlocks zu bewältigen.

Zur Einhaltung von **Corporate Standards** oder für Anwendungen, die auf mehreren Datenbanksystemen laufen, kann es notwendig sein, Transaktionen SERIALIZABLE zu entwickeln.

Transaktionen, die Fehler in der Serialisierbarkeit überprüfen und einen Neuanlauf starten, können den READ COMMITTED Modus benützen, da dieser keine Fehler in der Serialisierbarkeit erzeugt.

SERIALIZABLE sollte nicht die erste Wahl bei relativ langen Transaktionen die dieselben Zeilen verändern, die auch von einer hohen Anzahl von kurzen Transaktionen verändert werden. Da es sehr unwahrscheinlich ist, dass eine lange Transaktion die erste ist, die ein UPDATE auf eine Zeile durchführen will, wird es oft vorkommen, dass diese Transaktion mehrmals zurückgerollt wird und damit Arbeit vergeudet wird.

Zusätzlich sollten die Kosten für Zurückrollen und Neuanlauf/Starten für eine Transaktion beachtet werden, falls SEIALIZABLE Mode verwendet wird. Vor allem bei Read-Locking Systemen, bei denen es sehr häufig zu Deadlocks kommt, müssen fehlgeschlagene Transaktionen zurückgerollt und nochmals durchgeführt werden. In einer Umgebung, in der es oft zu Konkurrenz zwischen Transaktionen kommt, kann dies zu einer bedeutenden Verschwendung von Ressourcen führen.

In fast allen Umgebungen ist es sehr unwahrscheinlich, dass eine Transaktion, die zuvor einen Cannot serialize access Error erhielt, bei einem Neuanlauf nochmals mit einer anderen Transaktion einen Konflikt erzeugt. Aus diesem Grunde sollte man Statements, bei denen es leicht zu Konflikten mit anderen Transaktionen kommen kann, eher am Anfang einer Serializable Transaction durchführen. Natürlich ist auch das keine Garantie für das korrekte Beenden einer Transaktion. Darum sollte die Anwendung so programmiert sein, dass die Anzahl der Neuanläufe limitiert werden.

Obwohl der Serializable Mode von Oracle zu SQL92 kompatibel ist und im Vergleich zu Read-Locking Implementierungen einige Vorteile bietet, so zeigen sich doch einige kleinere Unterschiede in der Semantik. Anwendungsentwickler sollten also immer beachten, dass Reads in Oracle Writes nicht sperren, so wie es in manchen anderen Systemen üblich ist. Transaktionen, welche die Datenbankkonsistenz auf der Anwendungsebene überprüfen, benötigen deshalb spezielle Methoden, wie zum Beispiel die SELECT FOR UPDATE Anweisung. Vor allem bei Anwendungszugriffen auf Oracle von anderen Umgebungen sollte dies in Betracht gezogen werden.

3.4 Explizite Sperren

3.4.1 Übersicht

Die Locks teilen sich in folgende Hauptkategorien:

DML Locks (data locks)

DML Locks schützen die Daten. Zu den DML Locks gehören Table Locks, welche die gesamte Tabelle sperren, aber auch Row Locks, die nur bestimmte Zeilen sperren.

DDL Locks (dictionary locks)

DLL Locks schützen die Struktur von Schema-Objekten, wie zum Beispiel Definitionen von Tabellen und Views.

Internal Locks und Latches

Internal Locks und Latches schützen interne Datenbankstrukturen, wie zum Beispiel Datafiles und werden ausschließlich automatisch gesetzt.

3.4.2 DML Locks

Der Zweck von DML Locks ist die Erhaltung der Integrität von Daten, auf die gleichzeitig von mehreren Usern zugegriffen wird (Concurrency-Control). Sie schützen vor destruktiven Einflüssen von gleichzeitigen DML und/oder DDL Operationen.

Beispiel:

DML Locks garantieren dass eine bestimmte Zeile einer Tabelle nicht von mehreren Transaktionen gleichzeitig verändert werden kann oder eine Tabelle gelöscht wird, die ein noch nicht kommittiertes INSERT enthält.

DML Operationen können Locks auf zwei verschiedenen Stufen anfordern. Für einzelne Zeilen oder für ganze Tabellen.

Sperren auf Zeilen (Row Locks)

Die einzigen DML Locks, die Oracle automatisch anfordert, sind Sperren auf Zeilenebene. Es gibt keine Begrenzung für die Anzahl von Locks, die von einem Statement oder einer Transaktion angefordert werden können. Sperren auf Zeilenebene ermöglicht gezieltes Sperren und liefert daher beste Werte für Gleichzeitigkeit und Durchsatz.

Bei einer Kombination von Multiversion Concurrency Control und Locks auf Zeilenebene kommt es lediglich bei Usern, welche auf dieselben Daten zugreifen wollen, zu Konflikten.

Im Wesentlichen heißt das:

Lesend zugreifende User warten nicht auf schreibend zugreifende User derselben Zeile.
Schreibend zugreifende User warten nicht auf lesend zugreifende User derselben Zeile (außer der lesend zugreifende User verwendete die SELECT FOR UPDATE Anweisung, da diese einen Lock für den User anfordert).
Schreibend zugreifende User warten lediglich auf andere schreibend zugreifende User, falls sie versuchen dieselbe Zeile zur selben Zeit zu verändern.

Eine Transaktion fordert einen exklusiven DML-Lock für jede einzelne Zeile an, die von einem der folgenden Statements verändert wird:

INSERT
UPDATE
DELETE
SELECT in Verbindung mit der FOR UPDATE Klausel

Eine veränderte Zeile ist immer exklusiv gesperrt, sodass andere User sie nicht verändern können, bevor die Transaktion, die den Lock gesetzt hat, kommittiert oder zurückgerollt wird.

Falls eine Transaktion einen Lock auf eine Zeile bekommt, so erhält sie auch einen Lock auf die dazugehörige Tabelle. Der Lock auf die Tabelle verhindert DDL Operationen, welche die Änderungen einer laufenden Transaktion aufheben würden.

Sperren auf Tabellen (Table Locks)

Eine Transaktion erhält einen Lock auf eine Tabelle, falls diese mit einem der folgenden DML Statements verändert wird:

INSERT
UPDATE
DELETE
SELECT in Verbindung mit der FOR UPDATE Klausel
LOCK TABLE

Diese DML Operationen benötigen Locks auf Tabellen aus zwei Gründen:

Um DML Zugriffe auf die Tabelle für die Transaktion zu reservieren
und DDL Operationen zu verhindern, die einen Konflikt mit der Transaktion hervorrufen würden.

Jede Sperre auf eine Tabelle verhindert die Anforderung eines exklusiven DDL Locks auf dieselbe Tabelle und damit auch DDL Operationen, die einen solchen Lock benötigen würden.

Es kann daher zum Beispiel eine Tabelle nicht verändert oder gelöscht werden, solange eine noch nicht kommittierte Transaktion einen Lock auf die Tabelle gesetzt hat.

Ein Lock auf eine Tabelle kann in einem der folgenden Modi erfolgen:

row share (RS)
row exclusive (RX)
share (S)
share row exclusive (SRX)
exclusive (X)

Die Einschränkungen eines Table Lock Modus entscheiden, in welchen Modi andere Table Locks auf dieselbe Tabelle erreicht und gehalten werden können.

Die folgende Tabelle zeigt die Tabellensperrmodi, die von Statements benötigt werden und Operationen, die solche Locks erlauben oder verbieten.

SQL Statement	Mode of Table Lock	Lock Modes Permitted?				
		RS	RX	S	SRX	X
SELECT...FROM table...	none	Y	Y	Y	Y	Y
INSERT INTO table ...	RX	Y	Y	N	N	N
UPDATE table ...	RX	Y*	Y*	N	N	N
DELETE FROM table ...	RX	Y*	Y*	N	N	N
SELECT ... FROM table FOR UPDATE OF ...	RS	Y*	Y*	Y*	Y*	N
LOCK TABLE table IN ROW SHARE MODE	RS	Y	Y	Y	Y	N
LOCK TABLE table IN ROW EXCLUSIVE MODE	RX	Y	Y	N	N	N
LOCK TABLE table IN SHARE MODE	S	Y	N	Y	N	N
LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE	SRX	Y	N	N	N	N
LOCK TABLE table IN EXCLUSIVE MODE	X	N	N	N	N	N

RS: row share
RX: row exclusive
S: share
SRX: share row exclusive
X: exclusive
*Yes, if no conflicting row locks are held by another transaction; otherwise, waits occur.

Im Folgenden wird jeder Modus von Table-Locks gezeigt. Hierzu beginnen wir mit jenen, die die geringsten Einschränkungen treffen.

Row Share Table Locks

Zeigt an, dass die Transaktion, die den Lock auf die Tabelle gesetzt hat, auf Zeilen in der Tabelle einen Lock hält und diese zu verändern will. Dieser Lock wird automatisch angefordert, falls eines der folgenden SQL Statements ausgeführt wird:

SELECT ... FROM table ... FOR UPDATE OF ... ;
LOCK TABLE table IN ROW SHARE MODE;

Dies ist der Modus, der die wenigsten Einschränkungen bei einem Table-Lock setzt. Er bietet also den höchsten Grad an Concurrency für eine Tabelle.

Zulässige Operationen:

Er erlaubt anderen Transaktionen gleichzeitig das Abfragen, Einfügen, Verändern, Löschen oder Sperren von Zeilen der Tabelle. Daher können andere Transaktionen gleichzeitig ROW SHARE, ROW EXCLUSIVE, SHARE, und SHARE ROW EXCLUSIVE TABLE LOCKS auf dieselbe Tabelle erhalten.

Verbotene Operationen:

Er verhindert, dass andere Transaktionen exklusiv schreibend auf dieselbe Tabelle mit folgendem Statement zugreifen:

LOCK TABLE table IN EXCLUSIVE MODE;

Row Exclusive Table Locks

Die Transaktion, die den Lock gesetzt hat, hat mindestens eine Änderung in der Tabelle vorgenommen. Dieser Lock wird automatisch angefordert, falls eines der folgenden Statements auf die Tabelle ausgeführt wird:

INSERT INTO table ... ;
UPDATE table ... ;
DELETE FROM table ... ;
LOCK TABLE table IN ROW EXCLUSIVE MODE;

Dieser Lock setzt schon ein wenig mehr Einschränkungen, als der ROW SHARE Table Lock.

Zulässige Operationen:

Er erlaubt anderen Transaktionen das gleichzeitige Abfragen, Einfügen, Verändern, Löschen und Sperren von Zeilen der Tabelle. Daher erlauben Row Exclusive Table Locks mehreren Transaktionen, gleichzeitig Row Exclusive und Row Share Table Locks für dieselbe Tabelle zu setzten.

Verbotene Operationen:

Er verhindert, dass andere Transaktionen manuell Sperren auf die Tabelle setzten, um exklusiv lesend oder schreibend zuzugreifen. Daher können andere Transaktionen mit folgenden Statements nicht gleichzeitig Locks auf die Tabelle setzten:

LOCK TABLE table IN SHARE MODE;
LOCK TABLE table IN SHARE EXCLUSIVE MODE;
LOCK TABLE table IN EXCLUSIVE MODE;

Share Table Locks

Dieser Lock wird automatisch durch folgendes Statement angefordert:

LOCK TABLE table IN SHARE MODE;

Zulässige Operationen:

Ein Share Table Locks einer Transaktion erlaubt anderen Transaktionen nur das Abfragen der Tabelle, das Sperren von gewissen Zeilen durch SELECT ... FOR UPDATE und das Ausführen von LOCK TABLE ... IN SHARE MODE. Updates durch andere Transaktionen sind nicht erlaubt. Mehrere Transaktionen können gleichzeitig Share Table Locks für dieselbe Tabelle besitzen. In diesem Fall kann jedoch keine der beteiligten Transaktionen einen Update durchführen (selbst dann nicht, wenn eine Transaktion Row Locks in Folge eines SELECT ... FOR UPDATE Statements besitzt). Daher kann eine Transaktion, die einen Share Table Locks auf eine Tabelle besitzt, diese nur dann verändern, falls keine andere Transaktion ebenfalls einen Share Table Lock auf dieselbe Tabelle besitzt.

Verbotene Operationen:

ER verhindert, dass andere Transaktionen dieselbe Tabelle verändern oder eines der folgenden Statements ausführen:

LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE;
LOCK TABLE table IN EXCLUSIVE MODE;
LOCK TABLE table IN ROW EXCLUSIVE MODE;

Share Row Exclusive Table Locks

Dieser setzt bereits einige Einschränkungen mehr als ein Share Table Lock und wird durch folgendes Statement gesetzt:

LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE;

Zulässige Operationen:

Lediglich eine Transaktion kann zu einem Zeitpunkt einen Share Row Exclusive Table Lock auf eine Tabelle setzten. Dieser Lock erlaubt anderen Transaktionen das Abfragen sowie das Sperren von bestimmten Zeilen mit Hilfe des SELECT ... FOR UPDATE Statements, jedoch erlaubt er nicht das updaten der Tabelle.

Verbotene Operationen:

Er verhindert das Setzten von Row Exclusive Table Locks und das verändern derselben Tabelle durch andere Transaktionen. Dieser Lock verbietet außerdem anderen Transaktionen das Setzten von Share, Share Row Exclusive oder Exclusive Table Locks, die andere Transaktionen vom ausführen folgender Statements abhalten:

LOCK TABLE table IN SHARE MODE;

LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE;
LOCK TABLE table IN ROW EXCLUSIVE MODE;
LOCK TABLE table IN EXCLUSIVE MODE;

Exclusive Table Locks

Er setzt die meisten Einschränkungen für einen Table Lock und erlaubt der Transaktion, die ihn setzt, exklusiv schreibenden Zugriff auf die Tabelle. Angefordert wird eine Exclusive Table Lock wie folgt:

LOCK TABLE table IN EXCLUSIVE MODE;

Zulässige Operationen:

Lediglich eine Transaktion kann einen Exclusive Table Lock für eine Tabelle erhalten. Dieser Lock erlaubt anderen Transaktionen lediglich das Abfragen der Tabelle.

Verbotene Operationen:

Ein Exclusive Table Lock einer Transaktion verbietet es anderen Transaktionen, irgendwelche DML Statements auszuführen oder irgendeinen Lock auf die Tabelle zu setzten.

Zusammenfassung dieses Abschnitts:

DML Statement	Row Locks?	Mode of Table Lock
SELECT ... FROM table		
INSERT INTO table ...	X	RX
UPDATE table ...	X	RX
DELETE FROM table ...	X	RX
SELECT ... FROM table ... FOR UPDATE OF ...	X	RS
LOCK TABLE table IN ...		
ROW SHARE MODE		RS
ROW EXCLUSIVE MODE		RX
SHARE MODE		S
SHARE EXCLUSIVE MODE		SRX
EXCLUSIVE MODE		X

3.4.3 Default Locking für Abfragen

Abfragen sind die SQL Statements, die am wenigsten Einfluss auf andere SQL Statements haben, da sie nur lesend auf die Daten zugreifen. INSERT-, UPDATE- und DELETE-Statements können implizite Abfragen enthalten. Folgende Arten von Statements können Abfragen enthalten:

SELECT
INSERT ... SELECT ... ;
UPDATE ... ;
DELETE ... ;

Die folgenden Eigenschaften sind für alle Abfragen wahr, die keine FOR UPDATE Klausel benutzen:

Eine Abfrage setzt keine Data-Locks Daher können andere Transaktionen die Tabelle (einschließlich der abgefragten Zeilen) zur gleichen Zeit abfragen oder verändern. Da Abfragen, die keine FOR UPDATE Klausel enthalten, keine Data-Locks anfordern und daher andere Operationen nicht verhindern, werden solche Abfragen oft als **Nonblocking Queries** bezeichnet. D.h. eine Abfrage muss nicht auf die Aufhebung von Data-Locks warten, sondern kann jederzeit durchgeführt werden. (Nur in speziellen Fällen von verteilten Transaktionen muss gewartet werden.)

Default Locking für INSERT, UPDATE, DELETE, und SELECT ... FOR UPDATE

Die Sperreigenschaften für INSERT, UPDATE, DELETE und SELECT ... FOR UPDATE Statements sehen folgendermaßen aus:

Die Transaktion, die ein DML Statement enthält, fordert einen Exclusive Row Lock für die Zeilen an, die das Statement verändert. Andere Transaktionen können die gesperrten Zeilen weder verändern noch selbst sperren bis die erste (sperrende) Transaktion ihre Änderungen kommittiert oder zurückrollt.

Die Transaktion, die ein DML Statement enthält, muss keine Sperren auf die Zeilen anfordern, die von einer Abfrage oder einer Unterabfrage selektiert wurden. Eine Unterabfrage sowie eine implizite Abfrage innerhalb eines DML Statements sind konsistent im Bezug auf den Beginn der Abfrage. D. h. sie sehen keine der Änderungen, die das DML Statement vornimmt.

Eine Abfrage innerhalb einer Transaktion sieht die Änderungen, die von vorangegangenen DML Statements in derselben Transaktion durchgeführt wurden. Jedoch sieht sie nicht die Änderungen, die von anderen Transaktionen durchgeführt wurden, die nach der eigenen Transaktion begannen.

Zusätzlich zu den nötigen Exclusive Row Locks erhält eine Transaktion mit DML Statements auch mindestens einen ROW EXCLUSIVE Table Lock auf die Tabelle, welche die betroffenen Zeilen enthält. Falls die Transaktion schon einen Share, Share Row Exclusive oder einen Exclusive Table Lock auf die Tabelle hat, wird kein ROW EXCLUSIVE Table Lock benötigt. Falls die Transaktion bereits einen

ROW SHARE Table Lock besitzt, so wird dieser automatisch von Oracle in einen ROW EXCLUSIVE Table Lock umgewandelt.

3.4.4 DDL Locks (Dictionary Locks)

Ein DDL Lock schützt die Definitionen eines Schemaobjekts (z.B. Tabelle) während dieses Objekt von einer andauernden DDL Operation behandelt oder referenziert wird. (Ein DDL Statement kommittiert seine Transaktion implizit.)

Beispiel:

Ein User erzeugt eine Prozedur. Im Namen dieser Single-Statement-Transaktion des Users fordert Oracle automatisch DDL Locks für alle Schemaobjekte an, auf die in der Definition der Prozedur verwiesen wird. Diese DDL Locks verhindern, dass Objekte, auf die in der Prozedur verwiesen wird, verändert oder gelöscht werden, bevor die Übersetzung der Prozedur fertig ist.

Oracle fordert einen Dictionary Lock automatisch. Der User kann DDL Locks nicht explizit anfordern. Nur einzelne Schemaobjekte, die verändert oder auf die verwiesen wird, werden für die Dauer der DDL Operation gesperrt, jedoch nie das gesamte Data Dictionary.

DDL-Locks lassen sich in drei Kategorien einteilen:

exclusive DDL locks
share DDL locks
breakable parse locks

Exclusive DDL Locks

Die meisten DDL Operationen (ausgenommen Share DDL Locks) benötigen Exclusive DDL Locks auf eine Ressource um destruktive Einflüsse von anderen DDL Operationen, die dieselben Schemaobjekte verändern oder darauf verweisen wollen, zu verhindern. Einer DROP TABLE Operation ist es beispielsweise nicht erlaubt, die Tabelle zu entfernen, während eine ALTER TABLE Operation eine Spalte hinzufügt. Dies gilt natürlich auch umgekehrt.

Wird ein Schemaobjekt bereits durch einen DDL Lock gesichert, so muss eine andere Operation, die versucht einen Exclusive DDL Lock zu bekommen, warten bis der ältere DDL Lock freigegeben wird und kann erst dann fortsetzen.

DDL Operationen fordern ebenso DML Locks auf das Schemaobjekt an, welches verändert werden soll.

Share DDL Locks

Manche DDL Operationen benötigen Share DDL Locks auf eine Ressource um diese vor destruktiven Einflüssen anderer DDL Operationen zu schützen, erlauben aber Data Concurrency für gleichartige DDL Operationen.

Beispiel:

Falls eine CREATE PROCEDURE Anweisung ausgeführt wird, so fordert die Transaktion Share DDL Locks für alle Tabellen an, auf die darin verwiesen wird. Andere Transaktionen können ebenso gleichzeitig Prozeduren erzeugen, die auf dieselbe Tabelle verweisen und fordern somit ebenfalls Share DDL Locks auf die Tabellen an. Keine Transaktion kann einen Exclusive DDL Lock auf eine der betroffenen Tabellen erlangen. Keine Transaktion kann eine betroffene Tabelle verändern oder löschen. Es ist also jeder Transaktion, die einen Share DDL Lock besitzt, garantiert, dass die Definition der referenzierten Schemaobjekte für die Dauer der Transaktion konstant bleibt.

Ein Share DDL Lock auf ein Schemaobjekt wird für ein DDL Statement angefordert, dass eines der folgenden Kommandos enthält:

AUDIT, NOAUDIT, COMMENT, CREATE [OR REPLACE] VIEW/PROCEDURE/PACKAGE/PACKAGE BODY/FUNCTION/TRIGGER, CREATE SYNONYM und CREATE TABLE (falls der Parameter CLUSTER nicht verwendet wird)

Breakable Parse Locks

Ein SQL Statement (oder eine PL/SQL Programmeinheit) im Shared Pool besitzt einen Parse Lock für jedes Schemaobjekt, auf das es verweist. Parse Locks werden gesetzt um die verbundene SGA für ungültig zu erklären, falls ein referenziertes Objekt verändert oder gelöscht wird. Ein Parse Lock verhindert keine DLL Operationen und kann abgebrochen werden, um DLL Operationen, bei denen es zu Konflikten kam, möglich zu machen. Daher auch der Name Breakable Parse Lock.

Ein Parse Lock wird während der Analysephase (Parse Phase) der Ausführung von SQL Statement gesetzt und solange gehalten, solange sich die SGA für das Statement im Shared Pool befindet.

Die Dauer von DDL Locks

Die Dauer eines DDL Locks hängt von dessen Art ab. Exclusive und Share-DDL-Locks werden für die Dauer der Ausführung des DDL Statements und des automatischen Commits gehalten. Ein Parse Lock dauert genau so lange an, solange das damit verbundene SQL Statement im Shared Pool bleibt.

Latches und Internal Locks

Latches und Internal Locks schützen interne Datenbank- und Speicherstrukturen. Auf beide kann vom User nicht zugegriffen werden, da der User hier keine Kontrolle benötigt.

Latches

Latches sind Mechanismen auf Zeilenebene. Sie vereinfachen die Verwaltung interner Oracle-Operationen. Latches muss man sich als Sperren vorstellen, die im Millisekundenbereicherworben und freigegeben werden. Sie werden unmittelbar zum Anforderungszeitpunkt erworben und, falls die Datenbank ausgelastet ist, in Warteschlangen abgestellt.

Der Erwerb eines Latch ist nur ein Teil der Arbeit; bei Freigabe eines Latch gibt es auch eine Aufräumphase.

Latches benötigt man bei der Arbeit mit gemeinsam genutzten Strukturen in der SGA. Wird zur Arbeit mit einer bestimmten Struktur ein Latch benötigt und die Struktur ist belegt, wird eine Wait-Situation erzeugt. Es gibt zwei Latch-Typen:

- *Willing to wait*. Kann ein Latch nicht unmittelbar erworben werden, wird so lange eine Anforderung verschickt, bis sie erfüllt ist. Latches im Library Cache fallen beispielsweise in diese Kategorie.
- Anforderungen für *No-wait Latches* werden storniert, wenn das Latch nicht unmittelbar erhältlich ist. Die Anforderung wird neu gestellt. Zu dieser Kategorie gehören Latches auf Redo-Log-Kopien.

DBAs brauchen sich so lange nicht um Latches zu kümmern, bis sie, falls überhaupt, in der Datenbank zu Problemen führen.

Beispiele:

- Sie sollen gemeinsame Datenstrukturen in der SGA schützen.
- Latches schützen beispielsweise die Liste der derzeitig zugreifenden Datenbankuser und die Datenstrukturen, welche die Blöcke im Buffer Cache beschreiben.

Internal Locks

Internal Locks sind komplexere, auf höherer Ebene liegende Mechanismen als Latches und dienen mehreren Zwecken:

Dictionary Cache Locks

Diese Locks sind von sehr kurzer Dauer und werden auf Einträge im Dictionary Cache gehalten, während diese Einträge benutzt oder verändert werden. Sie garantieren, dass das analysierte Statement keine inkonsistenten Objektdefinitionen sieht.

Dictionary Cache Locks können Shared oder Exclusive sein. Shared Locks werden freigegeben, sobald die Analyse fertig ist. Exclusive Locks werden freigegeben, sobald die DDL Operation fertig ist.

File und Log Management Locks

Diese Locks schützen verschiedenste Files. Zum Beispiel schützt ein Lock das Control File, so dass es zu einem Zeitpunkt nur durch einen Prozess verändert werden kann. Ein anderer Lock koordiniert die Benützung und die Archivierung der Redo Log Files. Datafiles werden gesperrt um sicherzustellen, dass sich mehrere Instances mit einer Datenbank im Share Mode verbinden oder dass eine Instance sich mit dieser im Exclusive Mode verbindet. Da File- und Log Locks den Status der Files zu erkennen geben, müssen sie lange Zeit gehalten werden.

Hinweis: Besonders wichtig sind diese Locks, falls man den Oracle Parallel Server verwendet.

Tablespace und Rollback Segment Locks

Diese Locks schützen Tablespaces und Rollback Segmente. Es müssen beispielsweise alle Instances, die auf die Datenbank zugreifen damit einverstanden sein, ob eine Tablespace nun online oder offline ist.

Rollback Segmente werden gesperrt, sodass lediglich eine Instances in das Segment schreiben kann.

Oracle setzt Locks immer automatisch, um Data Concurrency, Data Integrity und Statement-Level Read Consistency zu sichern. Selbstverständlich kann der User diesen Default-Locking-Mechanismus außer Kraft setzen.

Dies ist vor allem in den folgenden Situationen nützlich:

Anwendungen benötigen Transaction-Level Read Consistency oder Repeatable Reads. Anders gesagt, Abfragen darin müssen konsistente Daten für die Dauer der Transaktion erzeugen und dürfen daher keine Änderungen von anderen Transaktionen sehen. Diese Lesekonsistenz auf Transaktionsebene kann der User erreichen, indem er entweder explizite Locks, Read-Only Transaktionen und Serializable Transaktionen verwendet, oder das Default Locking außer Kraft setzt (bzw. sich darüber hinwegsetzt).

Anwendungen benötigen exklusiven Zugriff auf eine Ressource durch eine Transaktion, sodass die Transaktion nicht auf andere Transaktionen warten muss um fertig gestellt zu werden.

Das automatische Setzten von Locks in Oracle kann auf zwei Ebenen außer Kraft gesetzt werden:

Transaktionsebene

Transaktionen, die eines der folgenden SQL Statements enthalten, setzten die automatische Anforderung von Locks in Oracle außer Kraft: SET TRANSACTION ISOLATION LEVEL, LOCK TABLE, SELECT ... FOR UPDATE. Locks die von solchen Statements angefordert werden, werden bei Commit oder Rollback wieder aufgehoben.

Sessionebene

Eine Session kann den benötigten Transaction Isolation Level mit dem ALTER SESSION Kommando setzen.

Sobald die Default Locks außer Kraft gesetzt werden, egal auf welcher Ebene, muss der Datenbank Administrator bzw. der Anwendungsentwickler sicherstellen, dass die neuen Locks richtig funktionieren. Diese Prozeduren müssen folgende Kriterien erfüllen:

Garantieren der Data Integrity
Möglichkeit von Data Concurrency
Unmöglichkeit von Deadlocks, bzw. entsprechende Behandlung von Deadlocks.

4. Instance

Der DBA ist für das Hochfahren und Herunterfahren der Datenbank verantwortlich

Schritte zum Hochfahren:

- Rufen Sie den Server Manager im Zeilenmodus oder SQL*PLUS auf.
- Geben Sie ein: CONNECT INTERNAL
- Fahren Sie die Datenbank hoch.
- Fahren Sie die Instance hoch.
- Mounten Sie die Datenbank
- öffnen Sie die Datenbank.

Frage: Wie geht das bei uns?

Schritte zum Herunterfahren:

Fahren Sie die Instance herunter um eine Datenbank für Benutzer unzugänglich zu machen.

Vorgang:

- Rufen Sie den Server Manager im Zeilenmodus oder SQL*PLUS auf.
- Geben Sie ein: CONNECT INTERNAL
- Fahren Sie die Instance herunter.

Frage: Wie geht das bei uns?

Bei vielen Betriebssystemen werden Skripts zum Hochfahren und Herunterfahren durch die Oracle Installationsprozedur eingerichtet.

Frage: Welche Bezeichung hat unser Skript?

Startup-Status

Startup Status	Beschreibung	Infos
Shutdown	Zustand als Ausgang	
NOMOUNT	CREATE DATABASE	init.ora wird gelesen, SGA aufgebaut, Hintergrundprozesse gestartet
MOUNT	Veränderungen der Dateistruktur, Archive Log-Modus einschalten, Recovery	verbindet DB mit Instance , öffnet Control-Dateien und vergleicht sie
OPEN	Macht Datenbank für alle Benutzer verfügbar	alle Redo-Log und Datendateien werden geöffnet

Zustand kann mit ALTER DATABASE geändert werden.

Beispiel:

```
STARTUP MOUNT
ALTER DATABASE OPEN
```

EXCLUSIVE erlaubt nur einer Instance den Zugriff
FORCE führt vor OPEN ein SHUTDOWN durch
RECOVER führt zunächst vollständiges Recovery durch

Datenbank herunterfahren:

Syntax	Beschreibung
SHUTDOWN ABORT	Nothalt, es wird kein Recovery durchgeführt, daher befindet sich Datenbank möglicherweise in inkonsistentem Zustand
SHUTDOWN IMMEDIATE	Ale aktuellen Befehle werden abgearbeitet, neue aber verhindert. Es erfolgt ein CLOSE und DISMOUNT der Datenbank, daher ist sie in konsistentem Zustand.
SHUTDOWN NORMAL	Auf angemeldete User wird gewartet, neu aber verhindert. Es erfolgt CLOSE und DISMOUNT

Hinweis: Zuerst wird immer die Instance gestartet und dann eine Datenbank dazugehängt.

Startup open pfile = path

5. Datenbankadministration

5.1 Datenbank erzeugen (Übersicht)

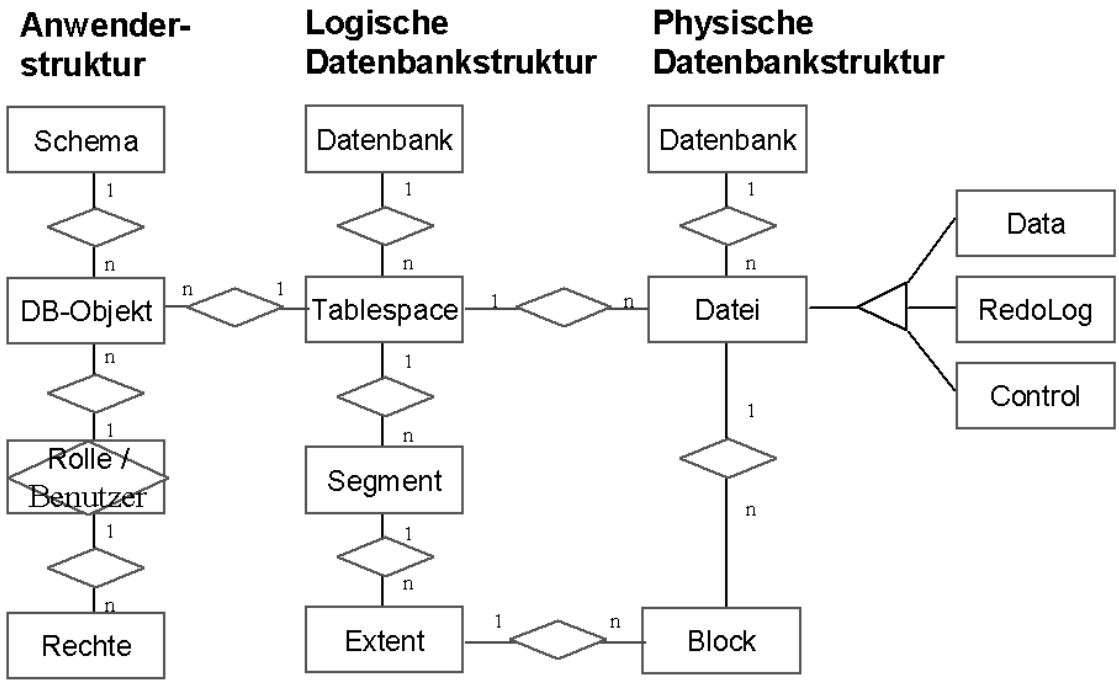
- Inhalte der Datenbank durch Tablespaces organisieren
- Datenbankstruktur entwerfen, um Zugriffskonflikte und Fragmentierung einzuschränken
- Betriebssystemumgebung für das Erzeugen der Datenbank vorbereiten
- Parameterdatei editieren
- Instance starten
  - Befehl CREATE DATABASE ausführen
  - Datenbanksicherheit durch Multiplex-Redo-Log-Dateien und Kontrolldateien sicherstellen
  - Passwortdatei erzeugen
  - Data Dictionary-Tabellen und Views definieren, um die Datenbank zu überwachen.

5.1.1 Datenbankstruktur von Oracle 8i (Übersicht)

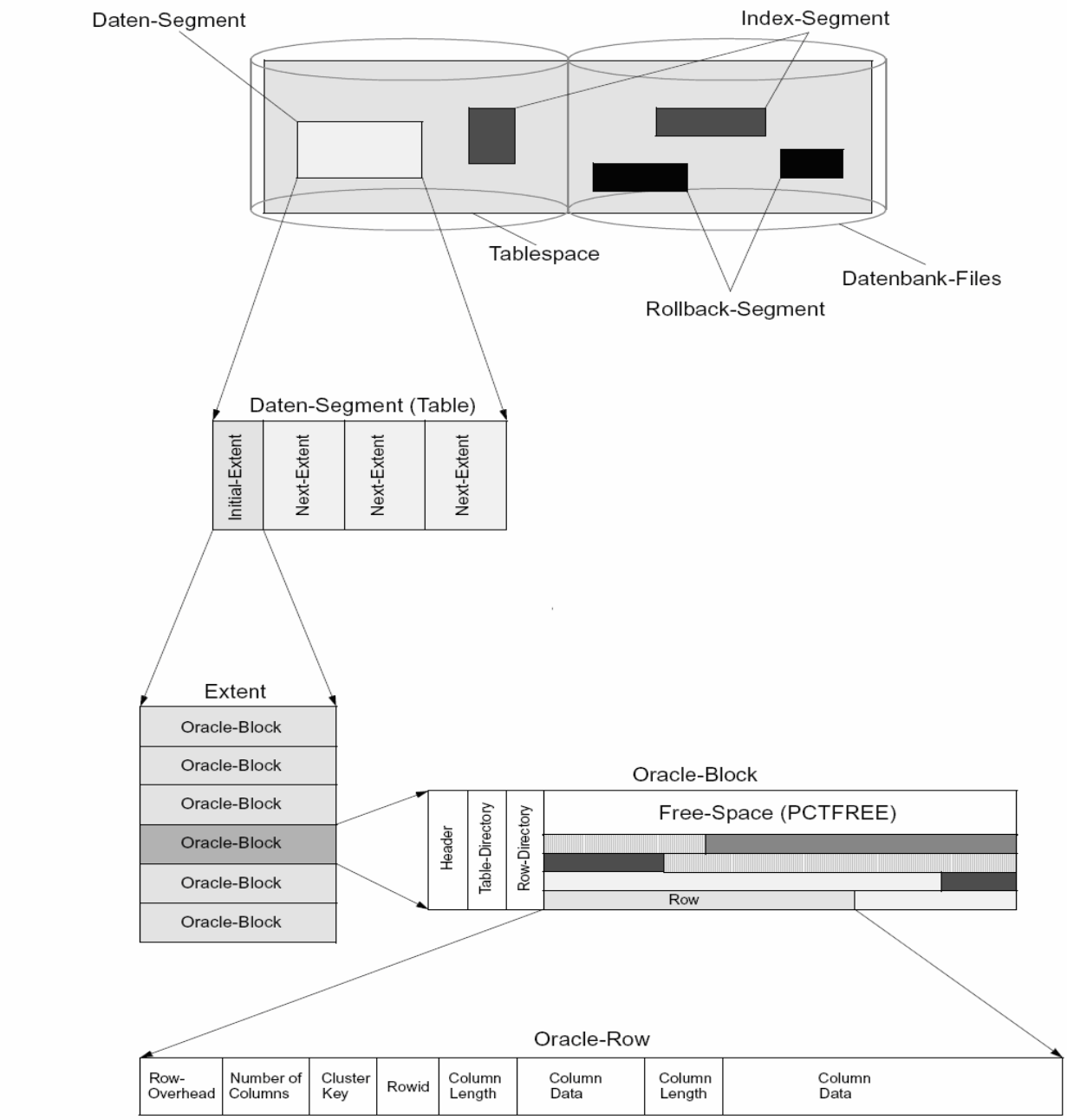
Wie wir die logische Struktur der Datenbank physisch abgebildet?

ER-Diagramm zeigt:

- Anwendungsstruktur
- logische DB-Struktur
- physische DB-Struktur



Übersicht Aufbau

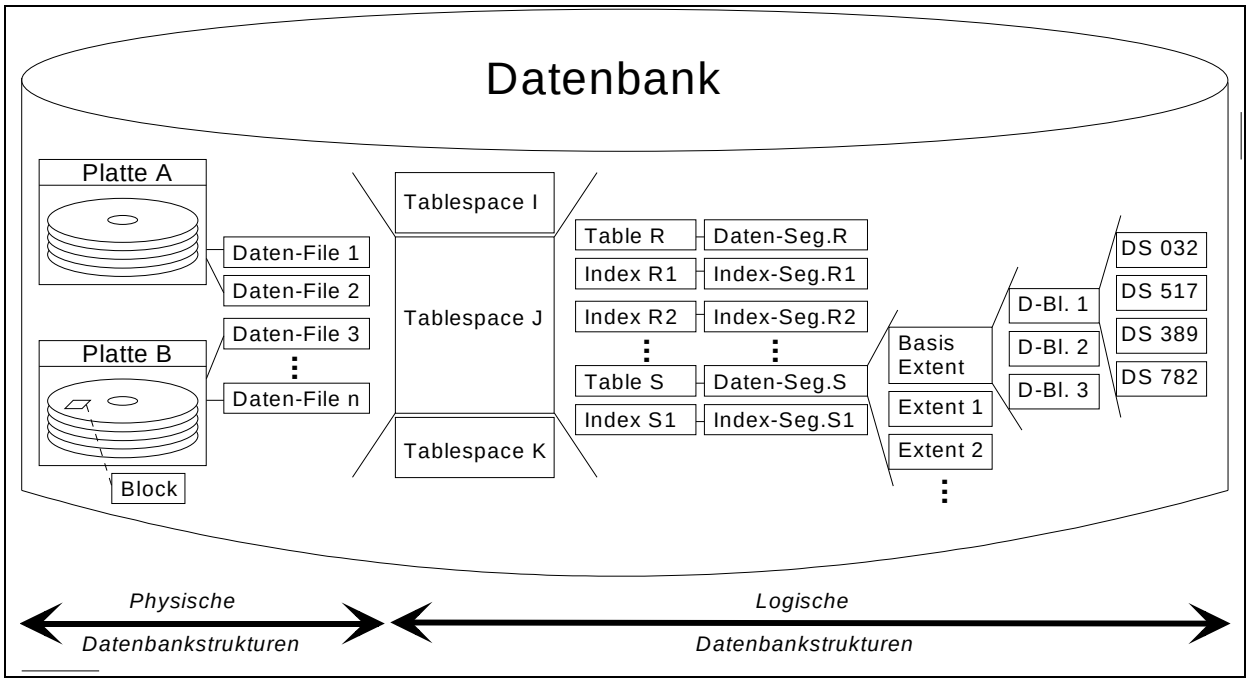


Hinweise:
Ein Tablespace kann sich über mehrere Datenbank Files erstrecken
Idealzustände: 1 Tablespace <-> 1 Datenbank-File.
1 Segment <-> 1 Extent.

Rows sollten nicht auf zwei Blöcke verteilt sein (Row-Chaining).

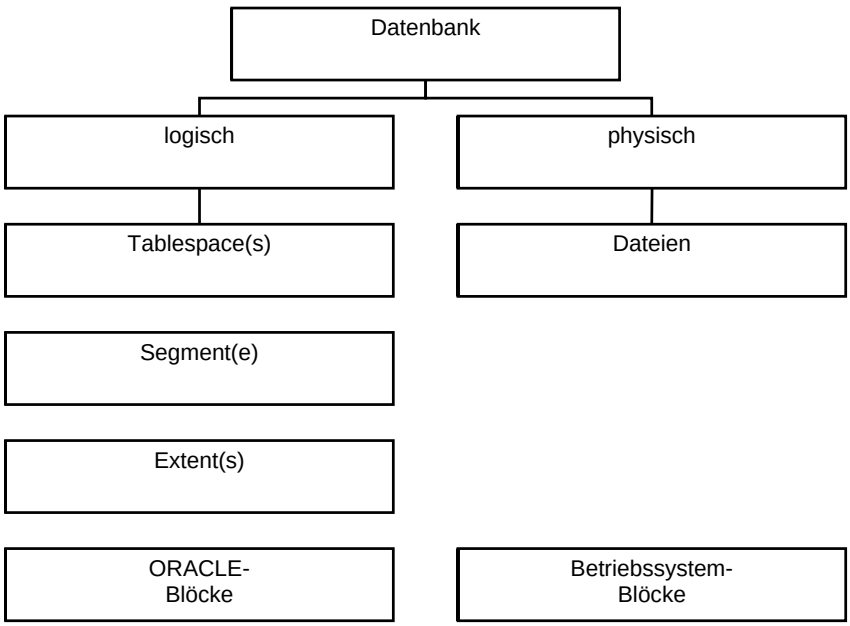
Oracle unterscheidet zwischen logischen Größen (Tablespace, Segmente etc.) und physikalischen Größen (Files, Blocks). Die physikalischen Größen werden beim Spezifizieren der logischen Größen angegeben (CREATE ...). Segmente sind Tables, Indexe, Rollback- und temporäre Segmente sowie Cluster. Oft wird dazu auch der Begriff Datenbank-Objekt benutzt.

Beispiel für Aufbau



5.1.2 Abbildung der logische Db-Struktur auf die physische Struktur

Übersicht



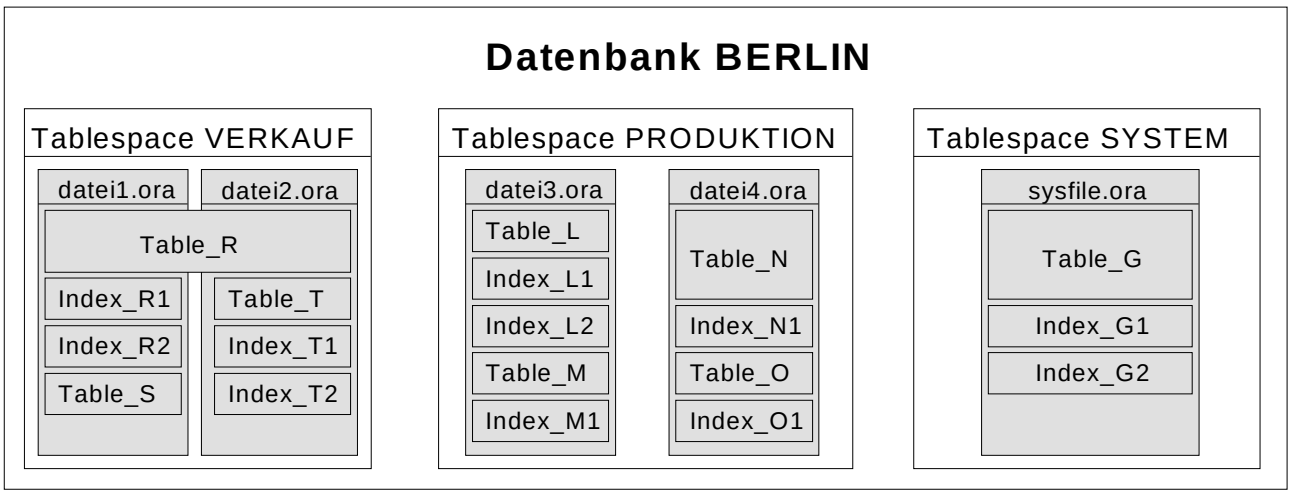
Für den DBA stellt sich die Datenbank in erster Linie als eine Menge von Dateien dar. Da Dateien physische, auf der Platte existierende Einheiten sind, kann diese Darstellung als physische Datenbankstruktur bezeichnet werden. Für die Anwender (Anwendungen) sollte jedoch kein Zwang bestehen, von den physischen Einheiten Kenntnis zu nehmen, damit es möglich ist, im Bedarfsfall die physische Struktur der Datenbank zu modifizieren, ohne dass gleichzeitig die Anwendungen geändert werden müssen.

Gemäß der ANSI/SPARC-Architektur wird deshalb die konzeptionelle Schicht geschaffen. Im Unterschied zur internen Sicht des DBA beruht sie nicht auf physischen Schichten, sondern auf logischen Einheiten. Diese Darstellung wird als Logische Datenbankstruktur bezeichnet.

Die physische Beschreibung der Datenbank beinhaltet alle Dateitypen: Daten-Dateien, Redo-Log-Dateien und Control-Dateien. Die logische Darstellung beschreibt eine alternative Strukturierung des Inhalts von Datendateien. Ein normaler Anwender braucht ja auch nichts von Redo-Log- und Control-Dateien zu wissen.

Die Datenbank

Beispiel:



Beispiel:

Database Prod												
Files Disk1/sys1.dbf		Disk2 User1.dbf			Disk3 User2.dbf			Disk1 roll1.dbf				Disk1 Temp.dbf
Tablespace System		USER_DATA						RBS				TEMP
Segments D.D. Table Data Seg	D.D. Index Index Seg	RB Seg	S-Dept Data Seg	S-Emp Data Seg	S-Dept Data Seg	S_Emp Ind Index Seg	Name	RBS1 RB Seg	RBS2 RB Seg	RBS1 RB Seg	RBS2 RB Seg	Temp Temp Seg
1 2 Extents	1 2	1 2	1	1 2	2	1	FREE	1	1	2	2	1
Oracle Database Blocks												

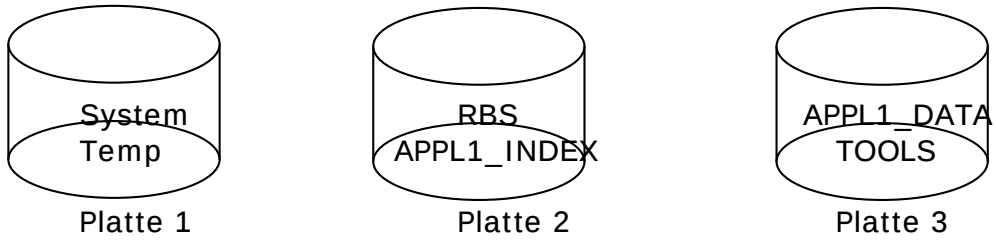
Eine Oracle Datenbank wird in kleinere logische Speicherbereiche unterteilt, genannt Tablespaces.

5.1.3 Tablespaces

- Jeder Tablespace besteht aus einer oder mehreren Betriebssystemdateien. Tablespaces können online gesetzt werden, während die Datenbank läuft.
- Mit Ausnahme des SYSTEM-Tablespace kann jeder Tablespace offline gesetzt werden während die Datenbank läuft.
- Jedes Objekt befindet sich nur in einem Tablespace. Tablespaces mit aktiven Rollback-Segmenten können nicht offline gesetzt werden.

Die Tablespaces können folgendermaßen aufgeteilt werden:

Beispiel:



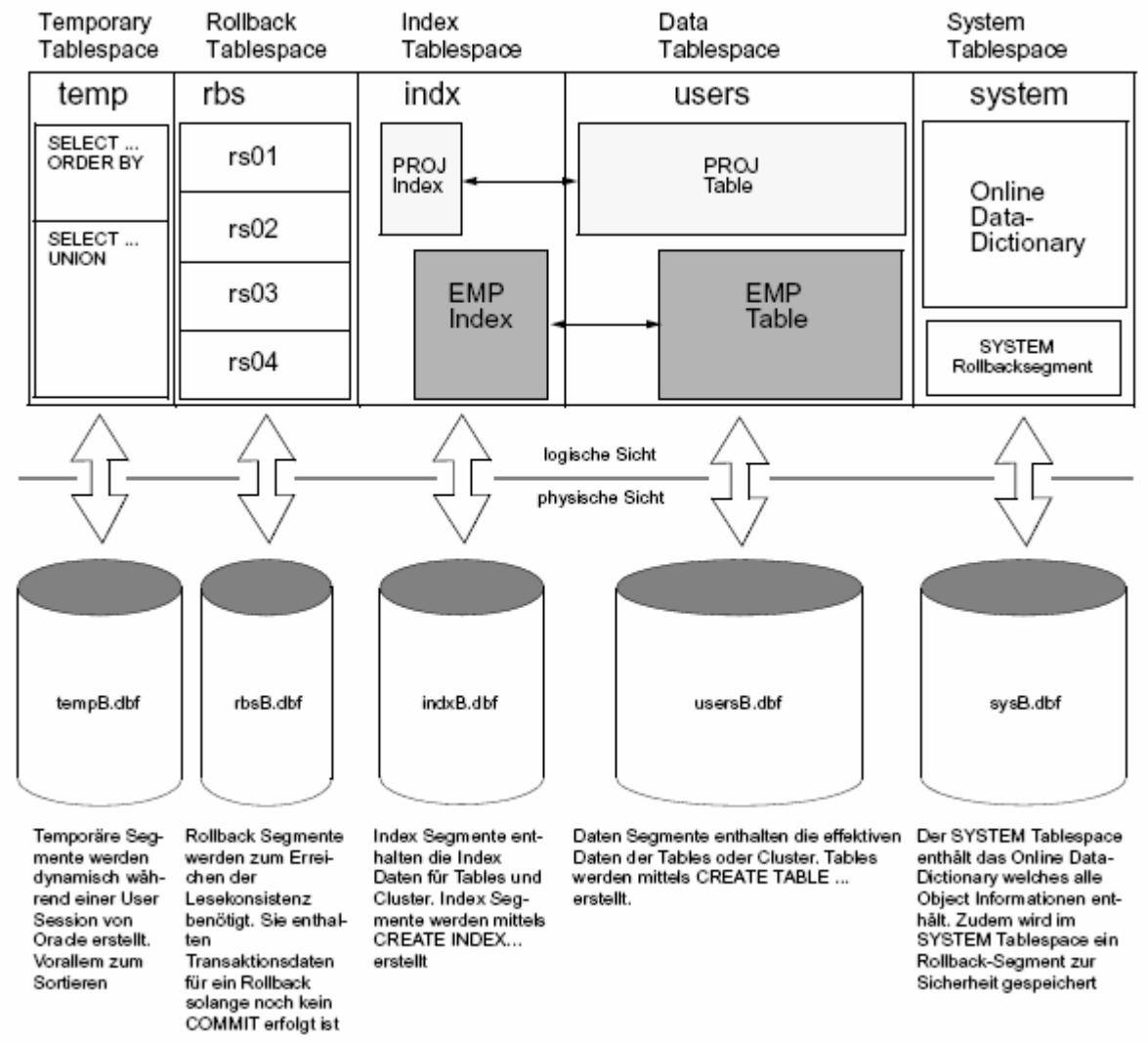
Die Tablespaces müssen so getrennt werden, dass

- wenig Fragmentierung erfolgt
- wenig Konflikte beim Zugriff auf Plattenspeicher erfolgen
- getrennte Segmente vorhanden sind

Es werden folgende Tablespaces angelegt:

Tablespace	Erforderlich	Beschreibung
SYSTEM	Immer	Alle Systeminformationen
TEMP	Optional	Speichert die temporären Segmente, die beim Sortieren erforderlich sind.
RBS	Optional	Speichert zusätzliche Rollback-Segmente. Ein Benutzer kann keinen Tablespace mit dem Namen rollback erzeugen, da ROLLBACK ein reserviertes Wort ist.
TOOLS	Optional	Sollte Tabellen enthalten, die von den ORACLE-Server-Tools verwendet werden (z.B. Enterprise-Manager).
APPL1_DATA	Optional	Enthält Produktionsdaten. Es kann mehr als einen Tablespace für die Produktionsdaten geben.
APPL1_INDEX	Optional	Enthält Indizes, die zu den Produktionsdaten im Tablespace APPL1_DATA gehören. Es kann mehr als einen Index-Tablespace geben.

Beispiel:



Tablespaces sollten auch getrennt werden, um unterschiedliche Fragmentierung zu vermeiden.

Fragmentierungsmerkmale:

Segment-Typ	Fragmentierungsmerkmal
Data-Dictionary-Segmente	Keine Fragmentierungstendenz und fragmentieren nie freien Speicherplatz
Anwendungs-Datensegmente	Geringe Fragmentierungstendenz, da Tabellen normalerweise die gleiche Lebensdauer besitzen wie das zugehörige Projekt
Temporäre Segmente mit Anwendungsdaten	Geringe Fragmentierungstendenz
Rollback-Segmente	Geringe Fragmentierungstendenz
Temporäre Segmente	Starke Fragmentierungstendenz

Weitere Eigenschaften, bei denen Segmente getrennt werden sollten:

- Bei unterschiedliche Backup-Erfordernissen
- Bei unterschiedlichen Zugriffserfordernissen
- Bei unterschiedlich erforderlichen Größen
- Mit unterschiedlicher täglicher Nutzung
- Mit unterschiedlicher Lebensdauer (Cleanup nach Ablauf eines Projektes)

Hinweis: Oracle hat fünf Arten von Segmenten, die vom Administrator verwaltet werden müssen. Cache-Segment muss nicht durch DBA verwaltet werden:

- **Daten -Segmente** enthalten die eigentlichen Daten der Tabellen. Jede mit einem CREATE - Table - Befehl erzeugte Tabelle hat genau ein Segment, in dem alle Daten der Tabelle gespeichert werden.

- **Index- Segmente** enthalten die Indizes auf Tabellen, die eine spezielle Zugriffsart beschleunigen sollen. In ORACLE sind B*-Bäume zum Aufbau von Indizes auf dem Primary - Key üblich und werden in den Index - Segmenten abgespeichert.

- **Rollback - Segmente** : Jede Datenbank enthält mindestens ein Rollback - Segment. Rollback - Segmente enthalten Informationen über alle noch nicht abgeschlossenen Transaktionen, auf die eine ROLLBACK erforderlich sein könnte. Sie sind zyklisch in der Art von Ringspeicherung aufgebaut. Rollback-Segmente werden bei Backup und Recovery und Zurücksetzen gescheiterter Transaktionen genutzt.

- **Temporäre Segmente** enthalten temporäre Daten, die z.B. bei der Ausführung eines SELECT - Befehls als Zwischentabellen oder zur Sortierung entstehen.

- **Cache-Segment** enthält Dictionary-Definitionen für Dictionary-Tabellen, die beim Öffnen der Datenbank geladen werden - muss nicht vom Datenbank-Administrator verwaltet werden.

5.2 Übersicht über das Erzeugen und Löschen einer Datenbank

5.2.1 Erzeugen einer Datenbank

Die Datei initorcl.ora ist zu kopieren (Name verschieden - .ora üblich)
- Instancenamen ändern
- Datenbanknamen ändern
- Blockgröße ändern etc.

Datenbank Zeichensatz
- MAX-Parameter einstellen

Die geeigneten Betriebssystemvariablen sind zu setzen

Servermanager/SQL-Plus aufrufen (orant\bin\svrmgr30 oder svrmgr oder svrmgrl)

```
STARTUP NOMOUNT pfile="C:\.......\initorcl.ora"
CREATE DATABASE (eventuell von Skript zur nachträglichen Kontrolle)
Svrmgr>@C:\.........\database.sql
SHUTDOWN ABORT
CONNECT INTERNAL/ORACLE AS SYSDBA
STARTUP OPEN pfile="C:\..........\initorcl.ora"
```

Ausführen der erforderlichen Skripts als mitgelieferte oder selbst erstellte SQL-Skripts.

Die wichtigsten sind im Verzeichnis RDBMS80\ADMIN zu finden.

Views zum überwachen der Datenbank

Größe der Datenbank:

```
compute sum of bytes on report
break on report
col bytes format 999,999,999,999
col tablespace_name heading 'Tablespace'
spool db_size.out
Select tablespace_name, sum(bytes) bytes
From dba_data_files
Group by tablespace_name;
```

Freien Platz bestimmen:

```
Select tablespace_name, sum(bytes) bytes
From dba_free_space
Group by tablespace_name;
```

Informationen über Tabellen:

```
Select Table_Name, Initial_Extent, Next_Extent,
Pct_Free, Pct_Increase
From dba_tables
```

```
Where Table_Name IN ('ORDER', 'CUST');
```

Informationen über Indizes:

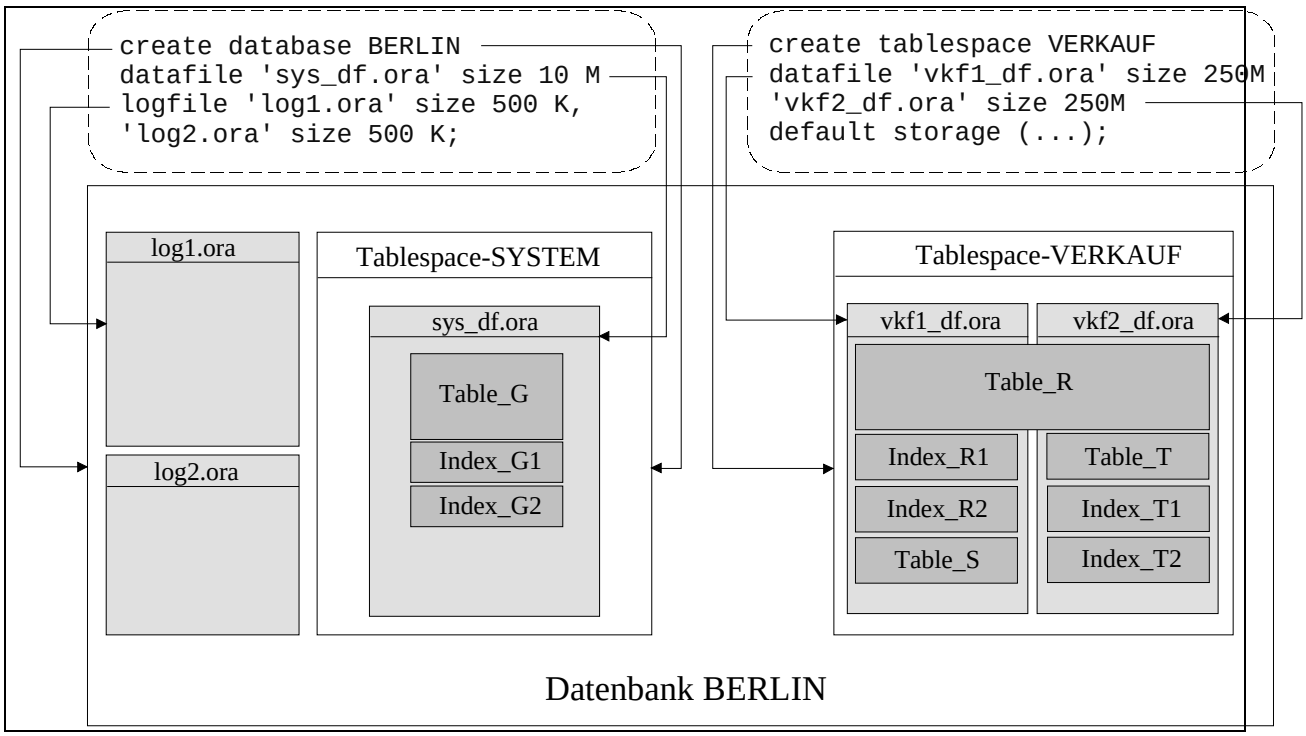
```
Select Index_name, Initial_Extent, Next_Extent,
Pct_Free, Pct_Increase
From dba_indexes
Where Index_Name IN ('ORDER_I', 'CUST_I');
```

Suche Tabellen/Indizes die in mehr als 5 Teile fragmentiert sind:

```
Select segment_name, extents, next_extent, bytes,
segment_type
From dba_segments
where extents > 4;
```

Lösung des Problems: Tabelle exportieren und reimportieren.

Beispiel:



Skript zum Erzeugen einer DB:

SPOOL: (Simultaneous Peripheral Operations On-Line).

```
spool d:\ORACLE\UEBSS3\Spool\inst_01.log

/*
 *   Starten einer leeren Instanz
 */
connect internal/DBMGR
```

```
startup nomount pfile=d:\ORACLE\ORA81\database\InitHDMV.ora

/*
 * Leere Datenbank anlegen
 *
 */

CREATE DATABASE UEBSS3
  LOGFILE GROUP 1
('d:\ORACLE\UEBSS3\OraData\RedoLogFilesA\RedoLogG01_1.dbf',
'd:\ORACLE\UEBSS3\OraData\RedoLogFilesB\RedoLogG01_2.dbf') SIZE 1M,
          GROUP 2
('d:\ORACLE\UEBSS3\OraData\RedoLogFilesA\RedoLogG02_1.dbf',
'd:\ORACLE\UEBSS3\OraData\RedoLogFilesB\RedoLogG02_2.dbf') SIZE 1M,
          GROUP 3
('d:\ORACLE\UEBSS3\OraData\RedoLogFilesA\RedoLogG03_1.dbf',
'd:\ORACLE\UEBSS3\OraData\RedoLogFilesB\RedoLogG03_2.dbf') SIZE 1M
DATAFILE 'd:\ORACLE\UEBSS3\OraData\SystemFile\sys01.dbf' size 100 M reuse
maxdatafiles 200
character set WE8ISO8859P1
noarchivelog;

/*
* Temporäres Rollbacksegment anlegen
*/
create rollback segment r0 tablespace system
storage (initial 16k next 16k minextents 2 maxextents 20);
alter rollback segment r0 online;

/*
 * Tablespaces für alle nicht explizit zugewiesenen User-Daten anlegen
 * dieser Tablespace muss von Zeit zu Zeit überprüft werden, ob Objekte
 * hier liegen. Diese müssen gegebenenfalls verlagert werden.
 */
create tablespace tsusr01
    datafile 'd:\ORACLE\UEBSS3\OraData\MiscData\UsrData_01.dbf' size 100M
reuse
  default storage
    (initial      100 K
     next         100 K
     pctincrease    0
     minextents     1
     maxextents   249
    )
;

/*
 * Tablespaces für temporäre Objekte, wird nicht von den Anwender direkt
 * genutzt.
 */
create tablespace TSTMP01
  datafile 'd:\ORACLE\UEBSS3\OraData\TempData\TempData_01.dbf'
  size     100 M
  REUSE
  default storage(
     initial      1 M
     next         1 M
     pctincrease    0
     minextents     2
     maxextents   249)
```

```
  TEMPORARY
;

/*
 * Tablespaces für Rollback-Segmente, wird nicht von den Anwender direkt
 * genutzt.
 */

create tablespace TSRLB01
  datafile 'd:\ORACLE\UEBSS3\OraData\RlbData\RlbData_01.dbf'
  size     60 M
  REUSE
  default storage(
     initial     512K
     next        512K
     pctincrease  0
     minextents   2
     maxextents 249);

/*
 * Rollback Segmente anlegen
 */
@d:\ORACLE\UEBSS3\create\rollback_segmente.sql

/*
 * Initial Rollback Segment wird vorläufig nicht gebraucht
 */
alter rollback segment r0 offline;


/*
 * Catalog-Files bringen eigene spools mit!
 */
spool off

/*
 * Catalog-Files für DataDictionary-Views, PL/SQL und Loader-Direct-Path
 */
@d:\ORACLE\ORA81\RDBMS\admin\catalog.sql
@d:\ORACLE\ORA81\RDBMS\admin\catproc.sql
@d:\ORACLE\ORA81\RDBMS\admin\catldr.sql

/*
 * System-Tabellen für User-Produktfile erstellen
 */
connect system/manager
@d:\ORACLE\ORA81\dbs\pupbld.sql
disconnect

/*
 * Weiter mit Anwendungsstrukturen, d.h. neues Spool-File
 */
spool d:\ORACLE\UEBSS3\Spool\inst_02.log

/*
 * User System default und temporary Tablespaces zuordnen und
 * dort unlimited quota zuweisen
 * Achtung: Vor der Installation von weiterer Oracle Software
 *          Default Tablespace auf tssystem setzen.
 */
```

```
connect internal/DBMGR
alter user sys    identified by SYBASE;
alter user system identified by DB2UDB;
alter user system temporary tablespace   tstmp01;
alter user system quota     unlimited on tstmp01;
alter user system default   tablespace   tsusr01;
alter user system quota     unlimited on tsusr01;

/*
 * Tablespace für die Anwendung TestApp anlegen
 */

 create tablespace TSUEBSS301
  datafile 'd:\ORACLE\UEBSS3\OraData\UEBSS3Data\UEBSS3_01.dbf'
  size     150 M
  REUSE
  default storage
    (initial      256K
     next         1 M
     pctincrease  0
     minextents   1
     maxextents 249
    );

/*
 * Benutzeranlegen
 * Vorlage normal user:
 *          create user xyz
 *            identified by abc
 *            default tablespace   TSUSR01
 *            temporary tablespace TSTMP01;
 *
 *            alter user xyz quota unlimited on tstmp01;
 *            alter user xyz quota unlimited on tsusr01;
 *
 *            grant connect to xyz;
 *
 */

create user MASTER
   identified by PROF
   default tablespace TSUEBSS301
   temporary tablespace TSTMP01;

alter user MASTER quota unlimited on tstmp01;
alter user MASTER quota unlimited on TSUEBSS301;
alter user MASTER quota unlimited on tsusr01;

grant connect to MASTER;
grant resource to MASTER;


/*
 * Anschliessend DB herunterfahren um den Archivelog-Modus einzuschalten
 *
 */

-- shutdown immediate
-- disconnect
-- connect internal/dbmgr
```

```
-- startup mount pfile=d:\orant\database\initHDMV.ora
-- Alter database archivelog;
-- Alter database open;
-- disconnect

spool off
```

Dazugehörige Rollbacksegmente erzeugen:

```
create rollback segment RLBSEG01 tablespace TSRLB01
storage (
          initial     512K
          next        512K
          optimal    1M
          minextents   2
          maxextents 249
        )
;
create rollback segment RLBSEG02 tablespace TSRLB01
storage (
          initial     512K
          next        512K
          optimal    1M
          minextents   2
          maxextents 249
        )
;
create rollback segment RLBSEG03 tablespace TSRLB01
storage (
          initial     512K
          next        512K
          optimal    1M
          minextents   2
          maxextents 249
        )
;
create rollback segment RLBSEG04 tablespace TSRLB01
storage (
          initial     512K
          next        512K
          optimal    1M
          minextents   2
          maxextents 249
        )
;
create rollback segment RLBSEG05 tablespace TSRLB01
storage (
          initial     512K
          next        512K
          optimal    1M
          minextents   2
          maxextents 249
        )
;
alter rollback segment RLBSEG01 online;
alter rollback segment RLBSEG02 online;
alter rollback segment RLBSEG03 online;
alter rollback segment RLBSEG04 online;
alter rollback segment RLBSEG05 online;
```

Parameter, die beim Erzeugen einer Datenbank angegeben werden sollten (andere Parameter werden üblicherweise, aber nicht zwingend geändert):

Parameter	Beschreibung
DB_NAME	Datenbank-Identifier mit höchstens 8 Zeichen. Beim Erzeugen einer neuen Datenbank muss nur dieser Parameter angegeben werden.
CONTROL_FILES	Name der Kontrolldateien (mindestens zwei). Datenbankname muss mit DB_NAME übereinstimmen.
DB_BLOCK_SIZE	Defaultwert ist 2048
SHARED_POOL_SIZE	Defaultwert ist 3.500.000
BACKGROUND_DUM_DEST	Speicherort für Tracedateien von Hintergrundprozessen
USER_DUMP_DEST	Speicherort für Benutzer-Tracedateien
DB_BLOCK_BUFFERS	Anzahl der Blöcke, die in der SGA gepuffert werden. Defaultwert ist 32
COMPATIBEL	Version des Servers, mit der diese Version kompatibel sein soll.

Nach dem Erzeugen durchzuführende Schritte zur Kontrolle:

Parametereinstellungen ausgeben

```
SHOW PARAMETER  -- Gespeichert sind diese in der V$PARAMETER
```

SGA-Größe abfragen

```
SHOW SGA        Gespeichert ist diese in der V$SGA
Hinweis: Kontrolle unter Linux: ipcs –m
```

System-Identifier angeben (SID)

Umgebungsvariable, die angibt, mit welcher Instance der ORACLE-Server einen Benutzer verbindet.

Beim Hochfahren der Instance sollte geprüft werden, ob die SID-Variable richtig eingestellt ist.

Beispiel: Setzen und überprüfen der SID

```
$ ORACLE_SID=TEST; export ORACLE_SID
$ echo $ORACLE_SID
TEST
```

Beim Anlegen einer DB sollte das CHAR-Set nicht vergessen werden. Dies ist sehr wichtig für den National-Language-Support bei verteilten Systemen.

CHAR-SET für Codepage 850: WE8ISO8859P1

Data Dictionary (Repository)

Data Dictionary

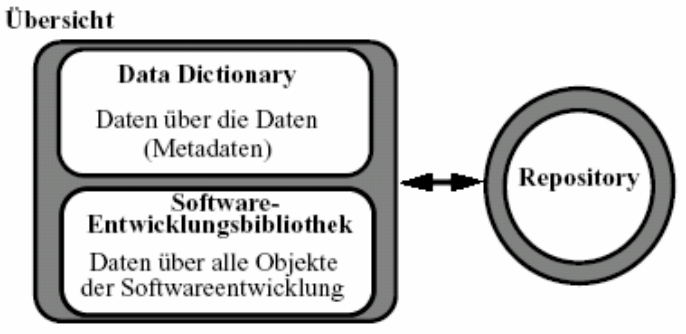
Verwaltung der Metadaten (also der Daten über die zu verwaltenden Daten) z.B. in DBMS

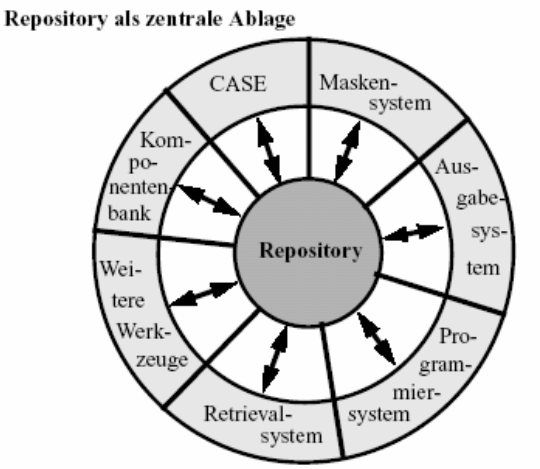
Software-Entwicklungsbibliothek

Werden anstelle der Daten alle Objekte verwaltet, die bei der Erstellung, Installation und Pflege von Software relevant sind (z.B. Programme, Module, Dateien usw.), spricht man vom Application Data Dictionary oder auch von der Software-Entwicklungsbibliothek

Repository

Der Begriff Repository wird benutzt, wenn man nicht mehr zwischen Data Dictionary und Software-Entwicklungsbibliothek unterscheiden möchte





Der Inhalt des Data-Dictionaries besteht im Wesentlichen aus zwei Gruppen:

Angaben über

- Definition
- Struktur
- Benutzung

der Daten

Beispiele:

- Namen
- Wertebereich
- Logische Beziehungen
- Integritätsregeln
- Zugriffsregeln

Angaben über

- Speicherung
- Codierung
- Auffinden

der Daten

Beispiele:

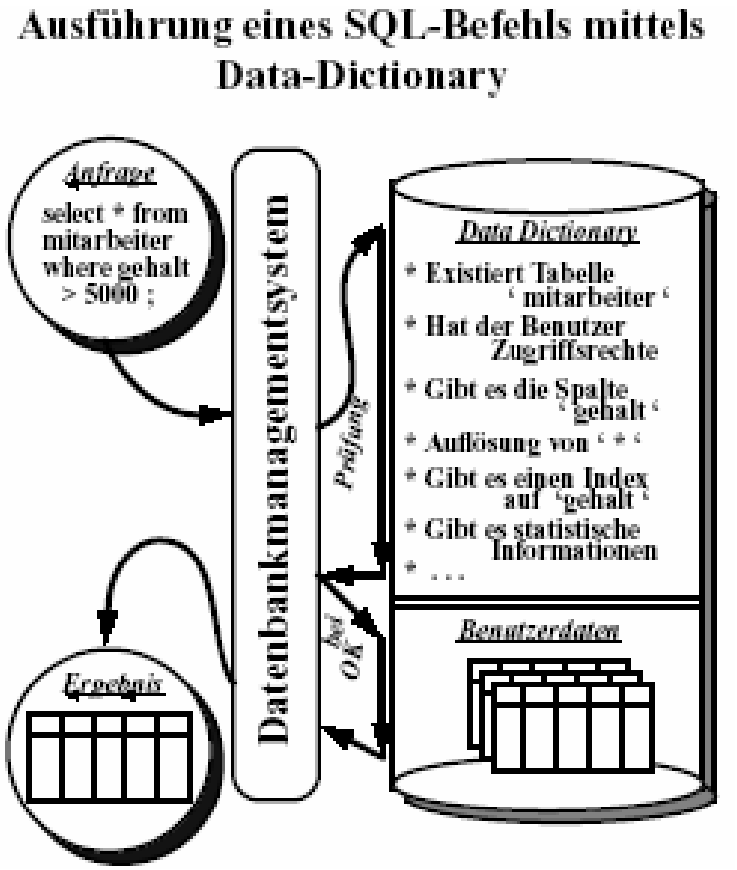
- Adressangaben
- Längenangaben
- Zugriffspfade
- Physische Platzierung
- Feldtypen

Wesentlicher Inhalt bei Oracle:

- Benutzernamen der ORACLE Server-Benutzer
- Privilegien und Rollen, die an die einzelnen Benutzer vergeben wurden
- Namen und Definitionen von Schema-Objekten
- Integritäts-Constraints
- Speicherplatz-Zuweisung für Datenbank-Objekte
- Allgemeine Datenbankstrukturen
- Audit-Informationen
- Stored Procedures und Datenbank-Trigger

Wer benutzt das Data Dictionary

Datenbankadministratoren
Datenbankbenutzer
Anwendungen
Der ORACLE-Server
Jeder Datenbank-Benutzer



Hinweis: Das Data Dictionary darf niemals durch einen Benutzer mit DML-Befehlen geändert werden!

Das Data Dictionary besteht aus einer Menge von Basistabellen und zugehörigen Views.

Kategorie	Beschreibung
USER_xxx	Informationen über die eigene Objekte für jeden Benutzer
ALL_xxx	Informationen über alle Objekte, die für einen Benutzer verfügbar sind (eigene Objekte plus die über GRANT erhaltenen Rechte).
DBA_xxx	Informationen für DBA, die alle Informationen über alle Objekte liefern.
ANSI-kompatibel	Synonyme, die der ANSI-Kompatibilität dienen, wie MYPRIVS oder ACCESSIBLE_TABLES
V$xxx	Dynamische View s, die während der Laufzeit geändert werden.

Wichtige VIEW

DICT -- Die Tabelle DICTIONARY enthält alle Tabellen des Data Dictionaries.

Beispiele (Übersicht):

Präfix / Postfix	USER_	ALL_	DBA_
TABLES Stellt alle Tabellen dar,	die der Benutzer angelegt hat.	auf die der Benutzer Zugriff hat	die im System verfügbar sind.
TAB_COLUMNS Stellt alle Spalten aller Tabellen dar,	die der Benutzer angelegt hat.	auf die der Benutzer Zugriff hat	die im System verfügbar sind.
INDEXES Stellt alle Idexes dar,	die der Benutzer angelegt hat.	auf die der Benutzer Zugriff hat	die im System verfügbar sind.
VIEWS Stellt alle Views dar,	die der Benutzer angelegt hat.	auf die der Benutzer Zugriff hat	die im System verfügbar sind.

5.2.2 Löschen einer Datenbank

Name aus der initorcl.ora löschen
Zugehörige physikalische Dateien löschen

5.3 Datenbankstruktur verwalten

Hinweis: Vergleiche dazu Übersicht über physische und logische Struktur der DB.

Dieses Kapitel umfasst:

Speicherbelegung
Datenbankstruktur anpassen
Tablespaces vorbereiten
Segmente richtig verwenden

Beschreibung der Datenbankstruktur - Übersicht:

Datenbank	logische Ansammlung von gemeinsam genutzten Daten, die in Tablespaces gespeichert werden
Datei	Physikalische Datendatei, die zu einem einzelnen Tablespace gehört.
Tablespace	Logisches Repository für physikalisch gruppierte Daten
Segment	Ein oder mehrere Extents, die Daten für eine bestimmte Struktur innerhalb eines Tablespace enthalten.
Extent	Zusammenhängende Datenblöcke in einer Datendatei
Block	Mehrere physikalische Dateiblöcke, die eine bestehende Datendatei belegt.

5.3.1 Tablespaces

- Jeder Tablespace besteht aus einer oder mehreren Betriebssystemdateien.
- Tablespaces können online gesetzt werden, während die Datenbank läuft.
- Mit Ausnahme des SYSTEM-Tablespace oder eines Tablespace mit einem aktiven Rollback-Segment kann jeder Tablespace offline gesetzt werden, während die Datenbank läuft.
- Tablespaces können zwischen dem Status Read-Write und Read-Only umschalten.
- In einem Tablespace erzeugte Objekte können nie Speicherplatz außerhalb ihres ursprünglichen Tablespace belegen.
- Jeder logische Tablespace besteht physikalisch aus ein oder mehreren Betriebsystemdateien
- Ein Segment, wie z.B. ein Datensegment, kann sich über mehrere Dateien erstrecken, solange diese Dateien zum gleichen Tablespace gehören.

Eine Datenbank besteht mindestens aus dem Tablespace SYSTEM. Weitere Tablespaces werden hinzugefügt, um Überwachung und langfristige Wartung zu vereinfachen.

Im Prinzip besteht für den Oracle Server die ganze Datenbank aus zwei Arten von Tablespaces: SYSTEM-Tablespace und Nicht-System-Tablspaces.

SYSTEM-Tablespace

- Wird bei allen Datenbanken für Datenbank-Operationen benötigt
- Enthält Data Dictionary-Informationen, Definitionen von Stored Procedures, Packages und Datenbank-Trigger
- Enthält das SYSTEM-Rollback-Segment
- Kann Benutzerdaten enthalten. Sollte es aber nicht.

Nicht-System-Tablespace

- Erlaubt mehr Flexibilität bei der Datenbankadministration
- Besteht aus Rollback-Segmenten, temporären Segmenten, Anwendungsdaten, Anwendungsindizes und Speicherplatz für die Benutzer

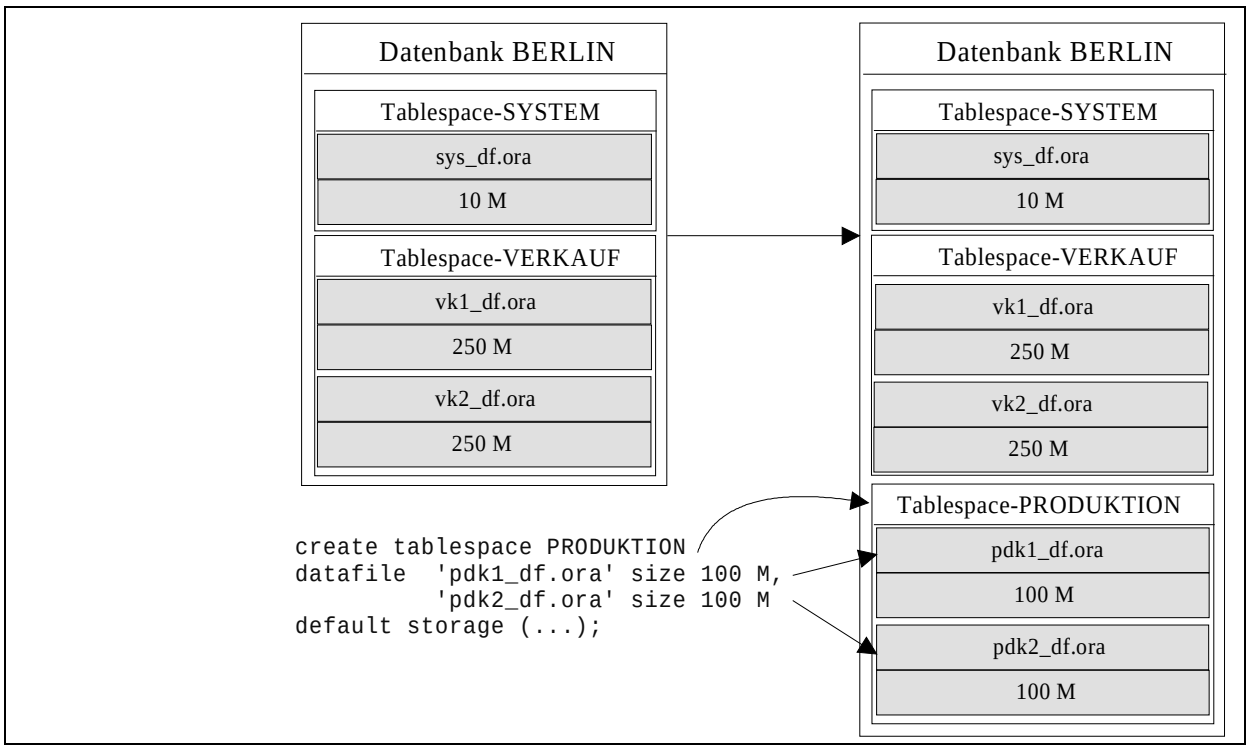
Ein Tablespace mit Rollback-Segmenten:

- Ist ein Nicht-System-Tablespace mit speziellen Eigenschaften
- Kann nicht offline gesetzt oder schreibgeschützt werden. wenn er ein aktives Rollback- Segment enthält. Er muss wie ein SYSTEM-Tablespace wiederhergestellt werden. d.h. indem die gesamte Datenbank für das Recovery offline gesetzt wird.

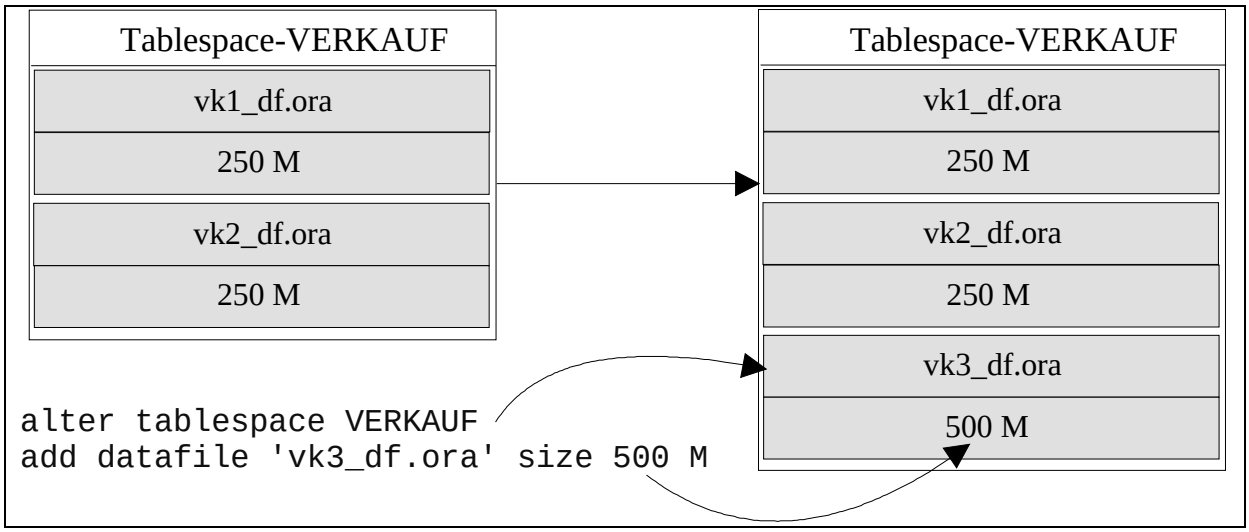
Anlegen eines Tablespace

Befehl: CREATE TABLESPACE

Beispiel: Anlage eines neuen Tablespace:



Beispiel: Hinzufügen von Dateien zu einem Tablespace:



Vergrößerung der Dateien eines Tablespace:

```
alter database datafile ‘vkf2_df.ora’ resize 500M
```

automatische Erweiterung der Dateien (unlimitiert) :

```
alter database datafile
      ‘vkf2_df.ora’ autoextend unlimited
```

bzw. automatische Erweiterung der Dateien (limitiert):

```
alter database datafile
      ‘vkf2_df.ora’ autoextend maxsize(500M)
```

Ein wichtiger Bestandteil bei der Erzeugung eines Tablespace ist die STORAGE-Klausel. Mit der STORAGE-Klausel lassen sich Speicherparameter vergeben, die die Speicherplatzzuordnung für Objekte steuern.

Die Datenbankobjekte werden als Segmente in der Datenbank abgelegt, wobei ein Segment aus einem oder mehreren Extents besteht. Ein Extent ist ein zusammenhängender Speicherbereich und setzt sich aus einer Anzahl von Datenbank-Blöcken zusammen.

Mit der Storage-Klausel lässt sich die Extentstruktur bezüglich Größe, Anzahl und Wachstum steuern.

Dabei kann die Steuerung der Extentstruktur von Segmenten auf:

Tablespace-Ebene

```
(CREATE/ALTER TABLESPACE DEFAULT STORAGE storage)
```

Objekt-Ebene

```
(CREATE/ALTER <object> STORAGE storage
```

Wenn ein Datenbank- Objekt wächst, wird Speicherplatz dafür angefordert.

Extents werden angefordert, wenn

- Das Segment erzeugt wird (INITIAL EXTENT)
- Das Segment wächst (NEXT EXTENT)
- Die Tabelle oder das Cluster geändert wird, um Extents anzufordern

Die Belegung von Extents wird aufgehoben, wenn

- Das Segment oder das Cluster gelöscht wird
- Das Segment oder das Cluster geleert wird (Truncate)
- Das Segment größer ist als OPTIMAL und noch freie Extents enthält (nur bei Rollback-Segmenten)

Merkmale von Extents:

- Jedes Segment in einer Datenbank wird mit mindestens einem Extent erzeugt das die Daten des Segments enthält
- Das erste Extent wird Initial-Extent genannt.
- Nachfolgende Extents werden Erweiterungs-Extents genannt.
- Ein Objekt belegt nur dann ein neues Extent, wenn seine aktuell belegten Extents bereits gefüllt sind.
- Häufiges Aufheben der Belegung von Extents kann zu einem fragmentierten Tablespace führen.
- Das INITIAL EXTENT ist vorbelegter Speicherplatz (chunk) innerhalb der Datenbank. Wenn das INITIAL EXTENT voll ist, wird das NEXT EXTENT belegt.
- Der Oracle Server betrachtet Blöcke mit aufeinanderfolgenden Block-ID s als zusammenhängend obwohl die Blöcke nicht notwendigerweise auf der Platte zusammenhängen müssen.

Speicherparameter in der Storage-Klausel

Parameter	Beschreibung
INITIAL	Größe in Bytes für das erste Extent, das für ein Segment belegt wird; der Default-Wert entspricht der Größe von fünf Blöcken.
NEXT	Größe in Bytes für nachfolgende Extents, die für ein Segment belegt werden; der Default-Wert entspricht der Größe von Fünf Datenblöcken
MAXEXTENTS	Gesamtzahl der Extents (einschließlich des ersten).die der Oracle Server für das Objekt belegen kann. Das Minimum ist l. Die Default-Werte hängen von der Datenblockgröße und dem Betriebssystem ab. Der größte Wert für MAXEXTENTS beträgt 121.
UNLIMITED	gibt an, dass Extents automatisch belegt werden, wenn sie benötigt werden. Sie sollten diese Option nicht für Rollback-Segmente oder Dictionary-Tabellen verwenden.

MINEXTENTS	Gesamtzahl der Extents, die belegt werden, wenn das Segment erzeugt wird. Default-Wert ist ein Extent, außer für Rollback-Segmente, die zwei benötigen.
PCTINCREASE	Prozentsatz, um den jedes nachfolgende Extent über das zuletzt belegte Erweiterungs-Extent wächst; Default-Wert ist 50 Prozent, außer für Rollback-Segmente, die diesen Parameter nicht verwenden
OPTIMAL	Optimale Größe in Bytes für ein Rollback-Segment. Default-Wert ist Null.
FREELISTS	Anzahl der Listen freier Blöcke, die für Eintragungen in eine Tabelle geführt werden; Default-Wert ist l.
FREELIST GROUPS	Anzahl getrennter Listen freier Blöcke, die von verschiedenen Instances eines Parallel Server verwendet werden; Default-Wert ist l

Hinweis: Durch Angabe von MINEXTENTS kann bei der Segment-Erstellung großer Speicherplatz belegt werden.

Vorrangregeln

- Ein Speicherparameter auf Objektebene setzt die entsprechende Option auf Tablespace-Ebene außer Kraft.
- Wenn Speicherparameter auf Objektebene nicht explizit gesetzt sind, werden die entsprechenden Default-Werte auf Tablespace-Ebene übernommen.
- Wenn Speicherparameter auf Tablespace-Ebene nicht explizit gesetzt sind, werden die entsprechenden Default-Werte des Oracle Server Systems übernommen.
- Werden Speicherparameter geändert dann gelten die neuen Optionen nur für Extents, die noch nicht belegt sind.
- Der Befehl OPTIMAL kann nur für Rollback-Segmente angegeben werden.
- Die Parameter FREELISTS und FREELIST GROUPS können nicht als Default-Werte für einen Tablespace angegeben werden.

NEXT und PCTINCREASE haben wesentliche Auswirkungen auf die Speicherplatzbelegung:

Auswirkungen von NEXT

NEXT gilt für den nächsten Erweiterungs-Extent Nachfolgende Extents können mit dem Faktor PCTINCREASE wie üblich wachsen.

Auswirkungen der Änderung von PCTINCREASE

Der Wert für NEXT wird wie folgt berechnet:

NEXT = NEXT * (1 + neue PCTINCREASE/100)

Auf den nächsten Oracle Block aufgerundet

Auswirkungen der Änderung von NEXT und PCTINCREASE

- Dem nächsten Extent wird unmittelbar der neue Wert von NEXT zugewiesen. Von diesem Punkt an wird NEXT wie üblich mit dem neuen Wert für PCTINCREASE berechnet.
- Änderungen der Tablespace-Default-Werte wirken nur für neue Objekte, die im Tablespace erzeugt werden.
- Die Parameter INITIAL und MINEXTENTS können nicht geändert werden.
- Diese Parameter können dazu dienen, die Performance durch Verringerung der DB-Größe zu erhöhen:
- INITIAL kann so groß gewählt werden, dass es für das ganze Segment ausreicht.
- PCTINCREASE kann so gewählt werden, dass Erweiterungs-Extents vergrößert werden.

Die Extent-Belegung kann mit folgendem Select angezeigt werden:

SELECT * FROM dba_free_space
WHERE tablespace_name = DATA
ORDER BY block_id;
Ein Tablespace kann nachträglich geändert werden mit ALTER TABLESPACE

Datendateien

Beim CREATE oder ALTER DATABASE werden auch die Datendateien angelegt.

Diese können: konstante Größe haben
automatisch vergrößert werden
in der Größe beschränkt werden
unbeschränkte Größe haben
umbenannt werden
verschoben werden

Tablespaces können auf READ ONLY gesetzt werden.

Freien Speicherplatz zusammenführen

Speicherplatz für Tablespace-Segmente werden über Extents verwaltet, die aus aufeinanderfolgenden Datenblöcken bestehen. Wenn ein größeres freies Extent fragmentiert ist, oder wenn es mehrere kleinere zusammenhängende Speicherplatz-Stücke gibt, kann dieser zusammengeführt werden. Die Zusammenführung wird defaultmäßig vom System Monitor Prozess (SMON) durchgeführt, kann aber auch mit dem SQL-Befehl ALTER TABLESPACE und der COALESCE-Klausel erreicht werden.

Beispiel:

```
SVRMGR> ALTER TABLESPACE users COALESCE;
```

In der View DBA_FREE_SPACE_COALESCED finden Sie statistische Informationen über die Extents eines Tablespace, die zusammengeführt werden können. Sie

können die View abfragen, um herauszufinden, ob ein Tablespace zusammengeführt werden sollte.

Nach jeweils acht Zusammenführungen führt die Speicherplatz-Transaktion einen Commit durch und andere Transaktionen können Speicherplatz belegen oder eine Belegung aufheben. Das bedeutet, dass Sie den Befehl ALTER TABLESPACE möglicherweise mehrmals aufrufen müssen. Fragen Sie die View DBA_FREE_SPACE_COALESCED ab.

Temporärer Tablespace

Innerhalb eines temporären Tablespace teilen sich alle Sortier-Operationen einer bestimmten Instance und Tablespace ein einzelnes Sortier-Segment. Für dieses gilt:

- Das Segment kann keine permanenten Objekte enthalten
- Der DBA verwaltet das Segment

Beim CREATE oder ALTER TABLESPACE kann dieser als

PERMANENT oder
TEMPORARY

angelegt werden.

Wichtige View: V$SORT_SEGMENTS

5.3.2 Richtlinien für die Verwaltung von Tablespaces

Es gibt drei Richtlinien für das Verwalten von Tablespaces:

1. Mehrere Tablespaces verwenden

- Die Verwendung von mehreren Tablespaces ermöglicht mehr Flexibilität bei der Durchführung von Datenbank-Operationen
- Benutzerdaten von Daten des Data Dictionary trennen
- Daten einer Anwendung von anderen Daten trennen
- Verschiedene Datendateien eines Tablespace auf verschiedenen Plattenlaufwerke speichern, um E/A-Zugriffskonflikte zu verringern
- Rollback-Segmente von Daten-Segmenten trennen, damit der Ausfall einer einzelnen Platte keinen vollständigen Datenverlust zur Folge haben kann
- Einzelne Tablespaces offline setzen, während andere online bleiben
- Tablespaces für bestimmte Verwendungszwecke reservieren, wie zum Beispiel hohe Änderungsaktivitäten. Read-Only-Aktivitäten oder Speicherung von temporären Segmenten
- Backup für einzelne Tablespaces durchführen

2. Speicherparameter für den Tablespace angeben

- Default-Speicherparameter für Objekte angeben, die im Tablespace erzeugt werden
- Default-Speicherparameter für einen Tablespace in Abhängigkeit von der erwarteten Größe der typischen Objekte dieses Tablespace setzen

3. Speicherplatz des Tablespace an Benutzer zuteilen

Speicherplatz des Tablespace den Datenbankbenutzern zuteilen

5.4 Segmente

Ein Segment ist eine Menge von ein oder mehreren Extents. Ein Segment enthält alle Daten für einen bestimmten logischen Speicherstruktur-Typ in einem Tablespace.

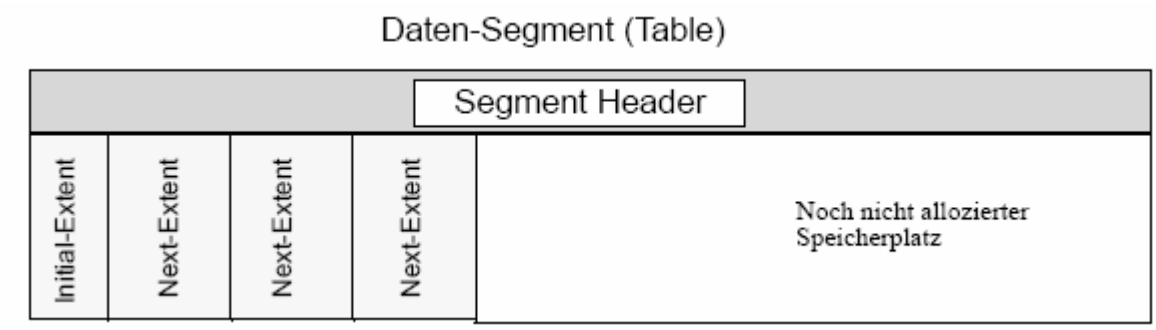
Segment-Typ	Beschreibung
Rollback-Segment	Ansammlung von Extents, die Daten für Rollback. Lesekonsistenz oder Recovery enthalten (Ring of Extents).
Daten-Segment	Ansammlung von Extents, die alle Daten für eine Tabelle oder ein Cluster enthalten.
Index-Segment	Ansammlung von Extents, die alle Daten für einen Index enthalten; Arbeitsbereich zum Sortieren von Daten (wird automatisch vom Server erzeugt, wenn kein Hauptspeicher für Sortiervorgänge verfügbar ist, und am Ende der Transaktion durch den SMON freigegeben).
Temporäres-Segment	Ansammlung von Extents, die Daten für temporäre Objekte enthalten (z.B. Sortieroperationen).
Cache-Segment/ Bootstrap-Segment	Extent, der Dictionary-Definitionen für Dictionary-Tabellen enthält, die beim Öffnen der Datenbank geladen werden; muss nicht vom Datenbank-Administrator verwaltet werden.

Was ist ein Segment?

- Ein Segment ist eine logische Struktur. Ein Segment kann erzeugt werden, Speicherplatz belegen und in der Größe wachsen. Segmente werden auch Datenbank- Objekte genannt.
- Segmente können sich nicht über mehrere Tablespaces erstrecken.
- Ein Extent ist eine Menge von zusammenhängenden Datenbankblöcken.
- Für temporäre Segmente gelten immer die Default-Speicherparameter des entsprechenden Tablespace. Sie können nicht explizit gesetzt werden.

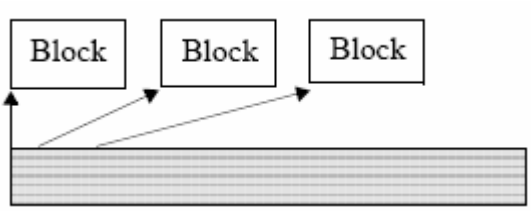
Aufbau der Segmente:

Der Aufbau der Segmente ist für alle gleich. Sie bestehen aus «Extents», welche wiederum in Blocks unterteilt sind. Oracle alloziert bei Bedarf immer ein zusätzliches Extent zu einer bestimmten Anzahl Oracle Blocks. Die Größe und Organisation der Extents wird beim Kreieren des Segments mittels «Storage Parameter» definiert.



Segment Header

Jedes Segment unterhält im Segment Header eine Liste der freien Blöcke. Beim Hinzufügen eines Extents werden alle Blocks dieser Liste zugefügt. Die Liste wird von Oracle verwaltet und ist nicht einsehbar. Bei INSERT's von Records muß diese Liste gelesen und modifiziert werden. Bei Tables mit sehr vielen Insert's kann dadurch ein Engpaß entstehen, was durch die dynamische Table V$WAITSTAT mit einem hohen Wert für buffer busy waits angezeigt wird. Man kann in diesem Fall den init.ora Wert FREE_LIST_PROC erhöhen. (Anzahl Free Lists per Instance).



Free-List im Segment-Header

Wichtige Views:

USER/DBA_EXTENTS
USER/DBA_FREE_SPACE
USER/DBA_SEGMENTS
DBA_TABLESPACES
DBA_DATA_FILES

Blöcke einer Tabelle berechnen:

```
select count(distinct(substr(rowid,1,8))) Anz_Blocks from help;

ANZ_BLOCKS
----------
       260
```

Anzahl der Zeilen pro Block:

```
select substr(rowid,1,8)Block_Nr, count(*) Anzahl_Rows
from help
group by substr(rowid,1,8);

BLOCK_NR ANZAHL_ROWS
-------- -----------
000005DF          14
```

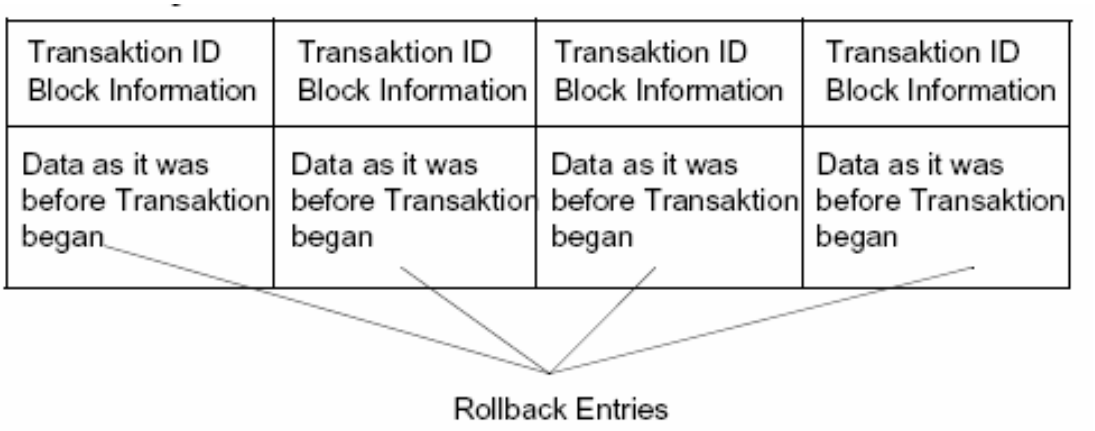
Verteilung der Rows pro Block:

```
select avg(count(*)) Avg, max(count(*)) Max, min(count(*)) Min
from help
group by substr(rowid,1,8);

 AVG        MAX        MIN
---- ---------- ----------
31.4         48          5
```

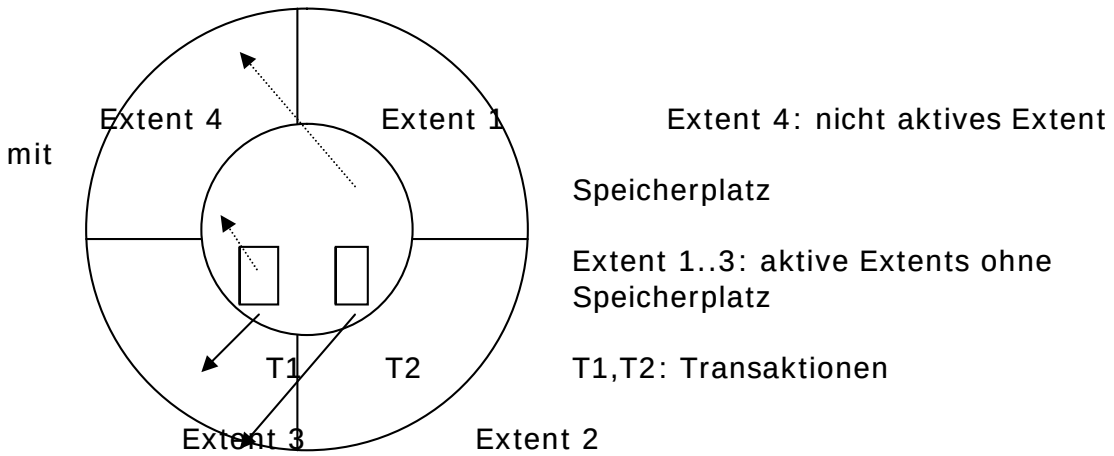
5.5 Rollbacksegmente verwalten

Rollbacksegmente enthalten die Information der «uncommited» Transaktionen, das heißt der Zustand der Daten vor Transaktionsbeginn. Oracle verteilt die aktiven Transaktionen auf die zur Verfügung stehenden Rollbacksegmente. Sind zuwenig Rollbacksegmente vorhanden so entstehen Waits. Zur Überprüfung kann die dynamische Table v$waitstat herangezogen werden. Der SQL *DBA Monitor kann zur Auswertung benutzt werden.



Ein Rollback-Segment besteht aus mehreren Extents. Die Extents eines Rollback-Segments enthalten Daten für Rollback von Transaktionen, Lesekonsistenz und Datenbank-Recovery.

- Sie sollten mehrere Rollback-Segmente verwenden.
- Es gibt public und private Rollback-Segmente.
- Die Größe der Rollback-Segmente muss geeignet gesetzt werden.
- Sie überwachen die Rollback-Segmente mit dem Server Manager.
- Für jedes Rollback-Segment wird eine OPTIMAL-Anzahl von Extents festgelegt.



Ein Rollback-Segment ist ein Teil der Datenbank, in dem Daten aufgezeichnet werden, bevor sie durch Transaktionen geändert werden. Dadurch können

Änderungen unter bestimmten Umständen zurückgerollt (rückgängig gemacht) werden.

Eine Transaktion ist eine Arbeitseinheit, die Änderungen auf der Datenbank ausführt oder Sperren auf Zeilen setzt. Jede Transaktion besitzt einen eindeutigen Identifier und ist genau einem Rollback-Segment zugewiesen. Ein Rollback-Segment wird zyklisch beschrieben, und eine Transaktion kann viele Einträge haben. Für eine Transaktion kann das nächste Extent verwendet werden, wenn dieses zurzeit keine aktiven Einträge enthält

Enthält das nächste zu verwendende Extent aktive Einträge, wird ein neues Extent zum Rollback-Segment hinzugefügt, sofern die maximale Anzahl von Extents noch nicht erreicht ist.

Eigenschaften von Rollback-Segmenten

- Erlauben im Fall von Benutzer-Fehlern das Zurückrollen der gesamten Transaktion bzw. bis zu einem Savepoint.
- Sichern die Lesekonsistenz
- Werden für das Datenbank-Recovery verwendet
- Können mit Ausnahme des SYSTEM-Rollback-Segments für jeden Tablespace bei Änderungen Daten aufzeichnen
- Können Änderungen für mehrere Transaktionen aufzeichnen
- Der SYSTEM-Tablespace enthält immer das Rollback-Segment SYSTEM.
- Ist mehr als ein Tablespace eingerichtet, ist mindestens ein zusätzliches Rollback- Segment erforderlich.

Einträge für Rollback-Segmente werden zyklisch geschrieben. Die Einträge werden für Rollback, Lesekonsistenz und Recovery verwendet.

Weitere Eigenschaften von Rollback-Segmenten:

- Enthalten Rollback-Einträge von mehreren Transaktionen
- Speichern Block-Informationen, wie Datei- und Block-ID, sowie Daten vor ihrer Änderung
- Können aufgrund einer großen Transaktion wachsen
- Können einer Transaktion automatisch oder explizit zugewiesen werden
- Können mit dem Befehl **ALTER ROLLBACK SEGMENT <rbs> SHRINK TO <size>** verkleinert werden.
- Verkleinern sich automatisch auf den OPTIMAL-Wert, falls sie erweitert wurden

Der Oracle Server benötigt ein online gesetztes SYSTEM-Rollback-Segment, wenn die Datenbank geöffnet ist.

Das SYSTEM-Rollback-Segment wird erzeugt, wenn die Datenbank angelegt wird.

Wenn eine Transaktion Rollback-Informationen in einem anderen Extent des Rollback- Segments weiterschreiben muss, vergleicht der Oracle Server die aktuelle Größe des Rollback-Segments mit der optimalen Größe des Segments. Wenn das Rollback-Segment größer als seine optimale Größe ist und die

unmittelbar auf das gerade gefüllte Extent folgenden Extents inaktiv sind, gibt der Oracle Server nicht aktive Extents des Rollback Segments frei. Er versucht die Freigabe, bis die Gesamtgröße des Rollback-Segments möglichst nah oder gleich der optimalen Größe ist, aber er verkleinert das Segment nicht unter die optimale Größe.

Eine Transaktion kann mit dem Befehl **SET TRANSACTION USE ROLLBACK SEGMENT** die Zuweisung eines bestimmten Rollback-Segments anfordern.

PCTINCREASE kann für Rollback-Segmente nicht angegeben werden.

Sie können den **OPTIMAL**-Wert abschätzen, wenn Sie V$ROLLSTAT überwachen oder den Befehl **MONITOR ROLLBACK** im Server Manager verwenden.

Speicherparameter für Rollback-Segmente

INITIAL
NEXT
MINEXTENTS
MAXEXTENTS
OPTIMAL

Parameterwerte

- OPTIMAL gibt die optimale Größe eines Rollback-Segments in Bytes an.
- Der Oracle Server gibt Extents frei, wenn ein Rollback-Segment größer als OPTIMAL ist.
- OPTIMAL darf nicht kleiner sein als die Gesamtgröße des ersten MINEXTENTS-Extents. Im Idealfall sollte die Größe gleich sein.
- PCTINCREASE kann für Rollback-Segmente nicht angegeben werden und ist immer auf Null gesetzt
- MINEXTENTS muss mindestens auf zwei gesetzt werden, da die zu einem Rollback-Segment gehörenden Extents fortlaufend und zyklisch verwendet werden.
- MAXEXTENTS sollte nie auf UNLIMITED gesetzt werden. da sonst die Gefahr besteht, dass der gesamte verfügbare Speicherplatz belegt würde.

Es gibt drei Arten von Rollback-Segmenten:

Typ	Beschreibung
privat	Muss mit dem Parameter ROLLBACK_SEGMENTS in der Parameterdatei angegeben werden, damit es vom System erkannt wird.
public	Pool aus Rollback-Segmenten, die von jeder Instance, die zusätzliche Rollback-Segmente benötigt, angefordert und verwendet werden können. Nur sinnvoll bei Umgebungen mit Oracle Parallel Server obwohl auch hier in der Regel private Rollback-Segmente bevorzugt werden.
deferred	Wird immer im SYSTEM-Tablespace erzeugt. Wenn ein Tablespace offline gesetzt wird, so dass Transaktionen nicht mehr sofort zurückgerollt werden können.

Richtlinien:

- Die bzw. der DBA kann nur public und private Rollback-Segmente erzeugen.
- Nur Benutzer mit dem CREATE ROLLBACK SEGMENT-Systemprivileg dürfen Rollback-Segmente erzeugen.
- Public Rollback-Segmente werden abhängig von TRANSACTIONS/TRANSACTIONS_PER_ROLLBACK_SEGMENT angefordert. Diese Berechnung bestimmt die Anzahl der Rollback-Segmente, die aus dem öffentlichen Pool entnommen werden.
- Public Rollback-Segmente werden manchmal beim Oracle Parallel Server verwendet. Sind aber nicht erforderlich. Normalerweise werden private Rollback-Segmente bevorzugt.

Erzeugen eines Rollback-Segmentes:

CREATE [PUBLIC] ROLLBACK SEGMENT name

Beim Erstellen und Tunen von Rollback-Segmenten sollten Sie zum Festlegen geeigneter Werte für die Parameter STORAGE und OPTIMAL SIZE die zu erwartenden Aktivitäten und die Anzahl der Transaktionen, die in ein Rollback-Segment schreiben werden, in Betracht ziehen.

Zugriffskonflikte bei Rollback-Segmenten

- Jede Transaktion wird einem Rollback-Segment zugewiesen.
- Es gibt nur eine endliche Anzahl von Header-Bereichen in einem Rollback-Segment, die für alle Transaktionen ausreichen müssen.
- Transaktionen, die keinen Header-Bereich erhalten, müssen warten.
- Je mehr Benutzer arbeiten, desto mehr Rollback-Segmente sollte es gehen.

Beispiel

```
SELECT n.name,round (100 * s.waits/s.gets)  %Count
FROM v$rollname n, v$rollstat s
WHERE n.usn = s.usn;
```

Wenn der Wert größer als 1 % ist, sollten Sie weitere Rollback-Segmente hinzufügen oder den Befehl SET TRANSACTION USE ROLLBACK SEGMENT verwenden.

Mit dem SQL-Befehl ALTER ROLLBACK SEGMENT oder CREATE ROLLBACK SEGMENT definieren Sie für ein Rollback-Segment die OPTIMAL-Größe in Bytes.

Merkmale der dynamischen Extent-Freigabe

- Ein Rollback-Segment kann eine optimale Größe besitzen. Die optimale Größe (in Bytes) wird angegeben, wenn das Segment erzeugt oder geändert wird.
- Ein Rollback-Segment wird automatisch auf die optimale Größe verkleinert, wenn die Anzahl der aktiven Transaktionen sinkt.

Richtlinien zur Bestimmung des OPTIMAL-Parameters

- Rollback-Segmente für lang andauernde Transaktionen sollten einen hohen OPTIMAL-Wert haben, um häufiges Zuweisen und Freigeben von Extents zu vermeiden.
- Bei Rollback-Segmenten für Transaktionen, die Informationen in langwierigen Abfragen aktualisieren, verwenden Sie einen großen OPTIMAL-Wert, um den Fehler snapshot too old (ORA-Ol555) zu vermeiden.
- Rollback-Segmente für kurz dauernde Update- und Abfrage-Transaktionen sollten einen kleinen OPTIMAL-Wert haben, um das Rollback-Segment-Caching zu erhöhen.
- Im MONITOR ROLLBACK des Server Manager oder durch Abfrage von V$ROLLSTAT können Sie sich über Zu- und Abnahme der Rollback-Segmente informieren. Eine genaue Angabe über die generierten Rollback-Informationen kann nur mit dem Befehl SELECT WRITES FROM V$ROLLSTAT vor und nach einer Datenbank-Operation bestimmt werden.

Rollback-Segment löschen:

Rollback-Segmente werden häufig gelöscht, weil die Extents des Segments zu stark fragmentiert sind oder das Segment auf einen anderen Tablespace verlagert werden muss. Man löscht ein Rollback-Segment mit dem SQL-Befehl DROP ROLLBACK SEGMENT.

Achtung! Das Rollback-Segment muss OFFLINE sein, damit es gelöscht werden kann.

6. Tabellen-, Index- und Clustersegmente

6.1 Tabellen verwalten

Segmente: Diese enthalten alle Daten einer bestimmten Struktur innerhalb eines Tablespace. Ein Segment besteht aus einem oder mehreren Extents.

Segment-Typ	Beschreibung
Daten	Enthält Daten
Index	Können verschiedene Arten sein, je nach Anforderung.

Tabellendaten werden als Zeilen gespeichert. Eine Zeile ist eine Kombination von horizontal angeordneten Werten einer Tabelle.

Eigenschaften:

Es gibt keine Begrenzung der Anzahl der Zeilen pro Tabelle

Eine nicht in einem Cluster abgelegte Zeile, die ganz in einem Block enthalten ist, hat einen mindestens drei Byte langen Zeilen-Header.

Zellen können sich über mehrere Blöcke erstrecken und sind dann verkettete Zeilen (chained rows).

Bei CHAR- oder DATE-Spalten werden die Zeilen fortlaufend gespeichert. Diese haben fixe Länge, daher kein PCTFREE (vgl. später).

Spaltenwerte werden direkt nacheinander gespeichert.

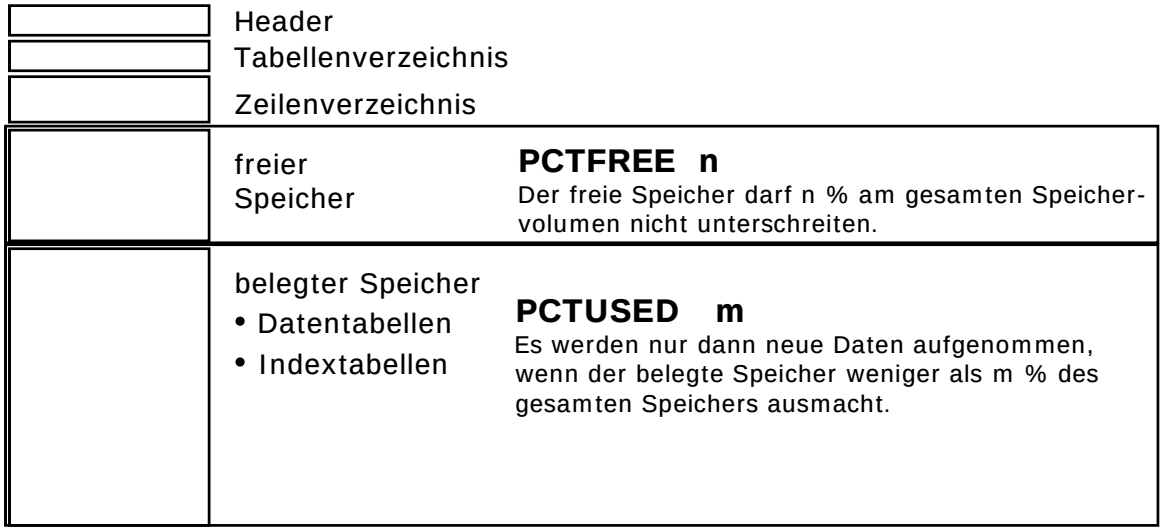
Die Spaltenlänge gibt die Anzahl von Bytes an, die zum Speichern des Spaltenwertes verwendet werden. Für die Längenangabe wird bei Längen bis zu 250 Bytes ein Byte und bei Längen größer als 250 Bytes werden drei Bytes benötigt.

Die Spaltenlänge 0 gibt an, dass das Feld NULL ist.

Hinweis: Für jede Zeile gibt es zwei Bytes im Zeilenverzeichnis des Datenblock-Headers.

Struktur eines DB-Blocks:

Gesamtblock:



Begriffe:

Header: Der Header enthält generelle Blockinformationen, wie Blockadresse, Type des Segments, (data, index oder rollback)

Table Directory: Das Table Directory enthält die Tabellennamen, die Tupel in diesem
Block gespeichert haben.

Row Directory: Das Row Directory enthält Informationen, welche Zeilen der betroffenen Tabellen tatsächlich gespeichert sind und unter welchen Adressen.

Free Space: Der Free Space wird reserviert zum Einfügen und Ändern von Datensätzen.

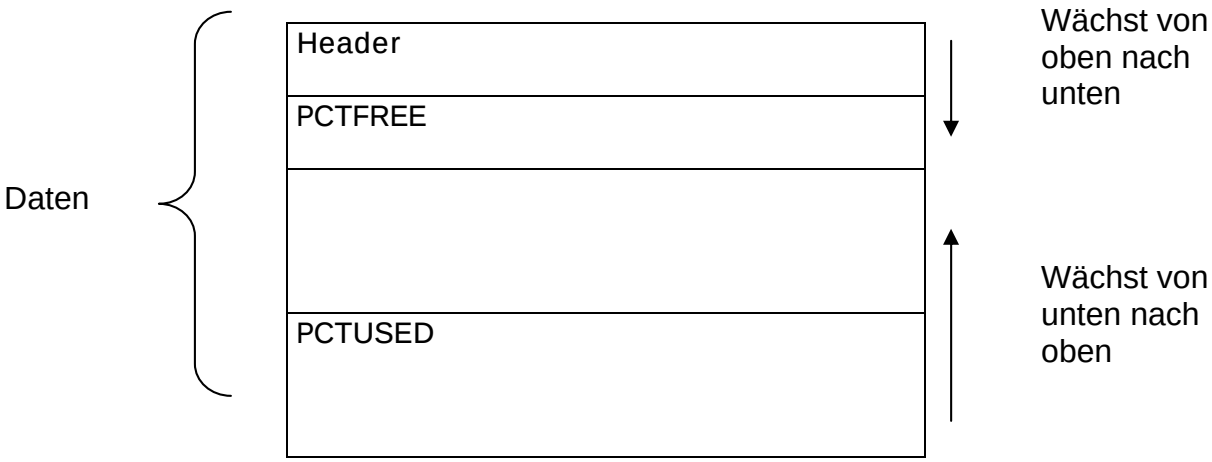
Row data: Der Row -data -Bereich enthält die eigentlichen Daten.

<table>
<tr><td colspan="5">Block-Header (Table-Directory, Row-Directory)</td></tr>
<tr><td>Zeilen-Header</td><td>Länge von Spalte 1</td><td>Daten für Spalte 1</td><td>Länge von Spalte 2</td><td>Daten für Spalte 2</td></tr>
<tr><td>Daten für Spalte 2</td><td>Länge von Spalte 3</td><td>Daten für Spalte 3</td><td>Länge von Spalte 4 NULL</td><td>Länge von Spalte 5</td></tr>
<tr><td colspan="3">Daten für Spalte 5</td><td>Länge von Spalte 6</td><td>Daten für Spalte 6</td></tr>
<tr><td colspan="4">Daten für Spalte 6</td><td>Pointer auf den Rest der Zeile</td></tr>
</table>

Verketteter Blockheader

Länge von Spalte 7	Daten für Spalte 7

Struktur eines Blocks



Struktur eines Block-Headers

Zeilen-Pointer	Zeilen-Pointer	Zeilen-Pointer	Zeilen-Pointer	Zeilen-Pointer
Zeilen-Pointer	Zeilen-Pointer			
	Transaktions-Eintrag	Transaktions-Eintrag	Transaktions-Eintrag	Transaktions-Eintrag

Das Resultat von richtig erzeugten Tabellen sind effizientere Datenbanken.

Speicherparameter

Sie speichern die ganze Tabelle, wenn möglich, in einem einzigen Extent.
Sie verwenden einen hohen Wert für PCTINCREASE, wenn die Tabelle noch entscheidend größer werden soll.
Sie verwenden viele kleine Extents mit einem geeigneten Wert für MINEXTENTS, wenn es keinen einzelnen freien Extent gibt.

Parameter zur Speicherplatzausnutzung

Sie setzen einen niedrigen Wert für PCTFREE, wenn die Zeilen nicht häufig verändert werden.
Sie setzen einen niedrigen Wert für PCTUSED, wenn häufig eingefügt und gelöscht wird.

PCTFREE plus PCTUSED darf l00% nicht überschreiten.

Wichtige Hinweise zum Anlegen einer Tabelle:

PCTFREE: Speicherplatz (als Prozentsatz des gesamten Blocks ohne Block-Header). der in jedem Block für länger werdende Zeilen reserviert wird
PCTUSED: anzustrebender minimaler Blockfüllgrad, der festlegt, wann ein Block für das Einfügen von Zeilen wieder zur Verfügung steht
INITRANS: Anzahl der Transaktionseinträge, die in jedem Block vorbelegt werden. Default-Wert ist 1.
MAXTRANS: Beschränkt die Anzahl der Transaktionseinträge, die in jedem Block belegt werden kennen. Default-Wert ist 255.
TABLESPACE: Tablespace, in dem die Tabelle erzeugt wird
STORAGE: Storage-Klausel, die festlegt wie Extents für die Tabelle belegt werden
RECOVERABLE: gibt an, dass die Erstellung der Tabelle im Redo Log protokolliert wird. Default-Einstellung = TRUE.
UNRECOVERABLE: gibt an, dass die Erstellung der Tabelle nicht im Redo Log protokolliert wird
CLUSTER: legt fest dass die Tabelle Teil eines Cluster wird
PARALLEL: gibt die Parallelitäts-Eigenschaften der Tabelle an. Weitere Informationen finden Sie bei der Beschreibung von DEGREE und INSTANCES.
DEGREE: ganze Zahl, die den Default-Grad der Parallelität für die Tabelle angibt. DEFAULT gibt an, dass der Parallelitätsgrad aus einer Schätzung der Tabellengröße und dem Wert des Initialisierungsparameters berechnet wird.
ENABLE: schaltet Integritäts-Constraints ein
DISABLE: schaltet Integritäts-Constraints aus
AS SUBQUERY: fügt die Ergebniszeilen der Unterabfrage in die Tabelle ein
CACHE: legt fest, dass bei einem Full Table Scan die für diese Tabelle gelesenen Blöcke im Cache-Puffer an den Kopf (most recently used) der LRU-Liste gestellt werden
N0CACHE: legt fest, dass bei einem Full Table Scan die für diese Tabelle gelesenen Blöcke im Cache-Puffer ans Ende (least recently used) der LRU-Liste gestellt werden

Die Storage-Klausel

INITIAL Größe des ersten Extent in Byte
NEXT Größe der nächsten Extents in Byte
MINEXTENTS Anzahl der zu Beginn angelegten Extents
MAXEXTENTS Gesamtzahl belegbarer Extents
PCTINCREASE Prozentsatz, um den jedes nachfolgende Extent über das zuletzt belegte Erweiterungs-Extent wächst
FREELISTS Anzahl der Freispeicherlisten
FREELISTGROUPS Anzahl getrennter Freispeicherlisten für Instances eines Parallel Servers

Mit **ALTER TABLE** können die Definitionen geändert werden. Zusätzlich können Werte hinzugefügt werden. Einige wichtige sind:

ALLOCATE EXTENT fügt einen neuen Extent manuell hinzu
SIZE Größe des neuen Extent in Byte
DATAFILE Name einer Datei des Tablespace
DEALLOCATE UNUSED gibt unbenutzten Speicherplatz am Ende der Tabelle frei und stellt ihn anderen Segmenten zur Verfügung
KEEP Anzahl Bytes oberhalb der High-Watermark nach Speicherfreigabe für die Tabelle

Beim Löschen einer Tabelle ist ganz besonders darauf zu achten, dass eventuell vorhandene Integritäts-Constraints mit CASCADE CONSTRAINTS mitgelöscht werden.

Es kommt häufiger vor, dass für ein Segment Speicherplatz belegt wird, der nicht genutzt wird. Die Belegung von nicht verwendetem Speicherplatz kann aufgehoben werden, damit andere Segmente den Platz nutzen können.

Innerhalb eines Segments zeigt die High-Watermark den verwendeten Speicherplatz an. Wenn Sie Zeilen löschen, wird die High-Watermark nicht verändert.

Die Belegung von Speicherplatz unterhalb der High-Watermark kann nicht aufgehoben werden.

Die High-Watermark kann mit der Package-Prozedur

DBMS_SPACE.UNUSED_SPACE angezeigt werden.

Wenn ein Segment vollständig leer ist, kann der Speicherplatz mit dem SQL-Befehl TRUNCATE DROP STORAGE freigegeben werden.

Die Klausel KEEP beim DEALLOCATE UNUSED ermöglicht, dass ein Teil (in Byte angeben) des Speicherplatzes nicht freigegeben wird.

Wichtige View:

DBA_FREE_SPACE (vgl. die Spalten EXTENT_ID, FILE_ID, BLOCK_ID)

Fragmentierter Speicherplatz kann, wenn er nicht von SMON zwingend zusammengelegt wird, über den Befehl

ALTER TABLE TABLESPACE users COALESCE;

Wichtige View:

DBA_FREE_SPACE_COALESCE

Datenbank-Blöcke

Ein Datenblock des Oracle Servers ist die kleinste von der Datenbank verwendete Speichereinheit.

Merkmale von Datenbank-Blöcken

- Datenbank-Blöcke entsprechen einem oder mehreren physikalischen Blöcken auf der Platte.
- Die Größe wird bei der Erzeugung der Datenbank mit dem Initialisierungsparameter DB_BLOCK_SIZE festgelegt und ist für alle Datendateien gleich.
- DB_BLOCK_SIZE legt ebenfalls die Größe der Datenbankpuffer in der SGA fest
- Datenbank-Blöcke werden auch als logische Blöcke oder als Oracle Server Blöcke bezeichnet.
- Wenn die Datenbank erzeugt ist, kann der Parameter DB_BLOCK_SIZE nicht mehr geändert werden.
- Die Größe für Oracle Server Datenbank-Blöcke ist typischerweise 2 KB, 4 KB oder 8 KB. Der Default-Wert hängt vom Betriebssystem ab.
- Die Größe der Oracle Server Blöcke sollte der Größe der Betriebssystemblöcke oder einem Vielfachen davon entsprechen.
- Af einigen Plattformen kann die Blockgroße sehr groß werden (zum Beispiel 32 KB). Diese Blöcke werden als Big Oracle Blöcke (Bob) bezeichnet.
- Alle E/A-Vorgänge werden auf Block bzw. Mehrblockebene durchgeführt. Der Oracle Server verwaltet Sperren auf Zeilenebene.

Bestandteile eines Datenbank-Blockes

Allgemeiner und variabler Header
Tabellenverzeichnis
Zeilenverzeichnis
Freier Speicherplatz

Achtung: Die Transaktionseinträge werden im freien Speicherplatz eingetragen und verbrauchen pro Eintrag 23 Byte.

Die wichtigsten Parameter für die Speicherplatzausnutzung sind:

PCTFREE (Default 10 %)
PCTUSED (Default 40 %)
INITRANS
MAXTRANS

Verkettung von Zeilen:

Eine Zeile, die nicht in einen Block passt, wird über mehrere Blöcke verkettet.

Migration

Eine Zeile wird migriert, wenn ein UPDATE mehr Speicherplatz benötigt, als aktuell im Block verfügbar ist. Eine migrierte Zeile wird vollständig vom ursprünglichen Block in einen verketteten Block verlagert. Der Zeilen-Header bleibt in dem Block, in dem die Zeile ursprünglich erzeugt wurde.

Die Analyse des Speichers:

Sobald COMPUTE STATISTICS eingeschaltet ist, wird Statistik über Datenbankereignisse (Indizes, Tabellen, Cluster) geführt (Performance!). Mit dem Befehl ANALYZE können später die Werte abgefragt werden.

6.2 Indizes

Wenn ein Index für eine Tabelle erzeugt wird, wird für den Index ein Index-Segment angelegt. Index-Segmente sind physikalisch von Tabellen- und Cluster-Datensegmenten getrennte Segmente.

Viele Indexmöglichkeiten, da wesentlicher Teil der Performance:

Wichtige Formen:

Unique Index vs. Non-Unique Index

Unique Index: zu jedem Indexeintrag gibt es nur einen Datensatz;
Non-Unique Index: zu jedem Indexeintrag gibt es beliebig viele Datensätze

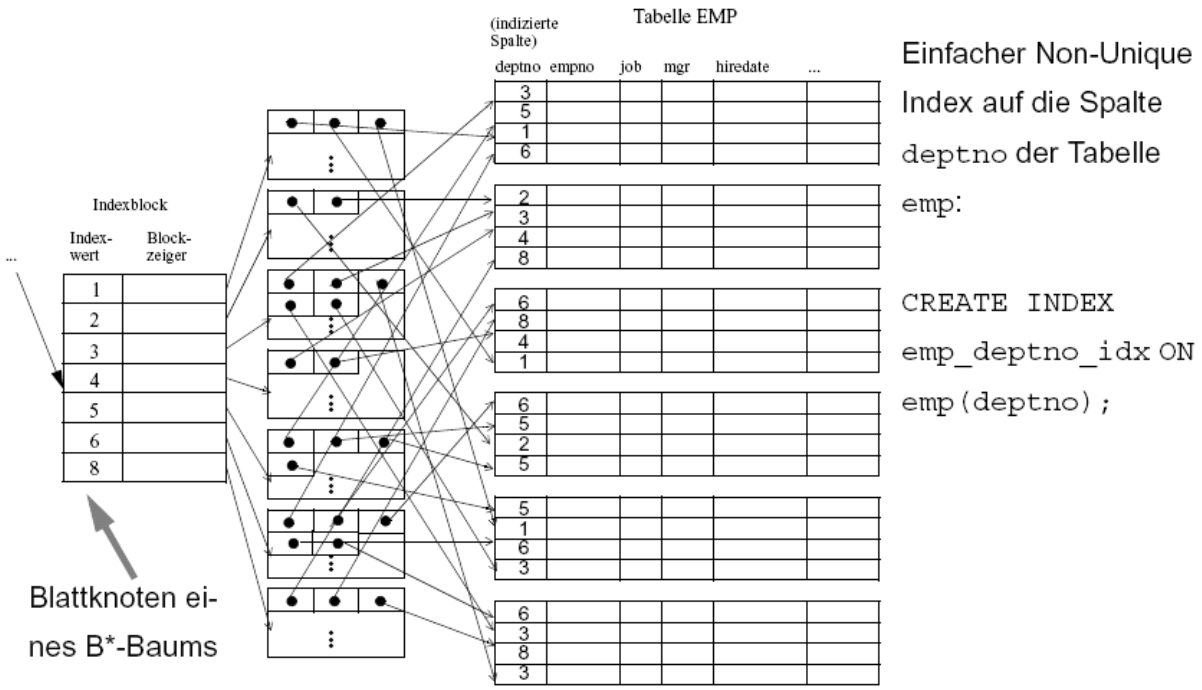
Ein Unique-Index wird automatisch von Oracle für Primärschlüssel (PRIMARY KEY-Constraint) sowie für Spalten mit UNIQUE-Constraint angelegt. Ansonsten nicht verwenden! Unique ist logisches Konzept!

Einfacher vs. Zusammengesetzter Index

Einfacher Index: Index ist auf einer Spalte definiert;
Zusammengesetzter Index: Index ist auf bis zu 16 Spalten definiert

Kandidaten für einen zusammengesetzten Index sind Spaltenkombinationen, die gemeinsam in WHERE-Klauseln vorkommen (AND-Verknüpfung).

Beispiel:



Syntax (Auszug):

```
CREATE INDEX   indexname          ON        tablename
               (
               column, ...
               column
               )
               TABLESPACE         tablespacename;

DROP   INDEX   indexname;

ALTER  INDEX   indexname  REBUILD;
```

Organisation der Lesezugriffe:

- Interne Formatierung der Daten-Files in Blöcke (512 Bytes bis 16 KB)
- 2 Zugriffsvarianten
  - Index-Zugriff ..... Lesen eines Blocks
  - Scan-Zugriff ..... Lesen von n-Blöcken (n...Parameter der Datenbank)
- Konsequenzen
  - Werden über einen Index-Zugriff viele Datensätze geladen, können bis zu n-mal mehr Lesezugriffe entstehen wie ohne Index!
  - Selektivität = (1-((N - distinct(N))/N))*100% > 80% entscheidet über die Güte
  - Spalten mit der höchsten Selektivität voranstellen

Ein Index unterstützt Anfragen mit

Wahlfreiem Zugriff (point query , multi-point query):
Suchen von Sätzen mit bestimmter Werteausprägung (= , Gleichverbund)

SELECT empno, job FROM emp WHERE deptno=2;

Indexsequentiellem Zugriff (range query):

Suchen v. Sätzen gemäß Sortierreihenfolge (< , > , BETWEEN, LIKE, ...)

SELECT empno, job FROM emp WHERE deptno BETWEEN 1 AND 5;

Index-Only Zugriff:

Abfrage selektiert nur Spalten die im Index liegen, d.h. kein Zugriff auf Datensegment nötig.
SELECT DISTINCT deptno FROM emp;

Indizes können angelegt werden als:

B*-Bäume
Bitmaps
Zusätzlich besteht die Möglichkeit, Tabellen als Hashing-Tables anzulegen.

Arbeitsweise eines Index

Select ohne und mit Index

ROWID	Name	Taetigkeit	Koennen
AAAAsYABQAAAAGzAAA	DICK JONES	SMITHY	Durchschnitt
AAAAsYABQAAAAGzAAB	JOHN PEARSON	COMBINE DRIVER	NULL
AAAAsYABQAAAAGzAAC	JOHN PEARSON	SMITHY	Durchschnitt
AAAAsYABQAAAAGzAAD	HELEN BRANDT	COMBINE DRIVER	Sehr schnell
AAAAsYABQAAAAGzAAE	JOHN PEARSON	WOODCUTTER	Gut
AAAAsYABQAAAAGzAAF	VICTORIA LYNN	SMITHY	Präzise
AAAAsYABQAAAAGzAAG	ADAH TALBOT	WORK	Gut
AAAAsYABQAAAAGzAAH	WILFRED LOWELL	DISCUS	Durchschnitt
AAAAsYABQAAAAGzAAI	ELBERT TALBOT	DISCUS	Langsam
AAAAsYABQAAAAGzAAJ	WILFRED LOWELL	WORK	Durchschnitt

Create Index I_WORKSKILL_1 on WORKSKILL (Name, Taetigkeit);

ADAH TALBOT	WORK	AAAAsYABQAAAAGzAAG
DICK JONES	WORK	AAAAsYABQAAAAGzAAA
ELBERT TALBOT	SMITHY	AAAAsYABQAAAAGzAAI
HELEN BRANDT	DISCUS	AAAAsYABQAAAAGzAAD
JOHN PEARSON	COMBINE DRIVER	AAAAsYABQAAAAGzAAB
JOHN PEARSON	SMITHY	AAAAsYABQAAAAGzAAC
JOHN PEARSON	WOODCUTTER	AAAAsYABQAAAAGzAAE
VICTORIA LYNN	WOODCUTTER	AAAAsYABQAAAAGzAAF
WILFRED LOWELL	DISCUS	AAAAsYABQAAAAGzAAH
WILFRED LOWELL	WORK	AAAAsYABQAAAAGzAAJ

Select mit und ohne Index

- Iterator Select
  - OPEN Table
  - DO UNTIL (EOF Reached)
    - Get Data from Table
    - IF (Condition) Store Data in Temp
  - DO NEXT

Lesen aller Datensätze

- Iterator IndexSelect
  - OPEN Index
  - Position According to Condition
  - OPEN Table
  - DO UNTIL (Condition)
    - Get Data from Table using RowID from Index
    - Store Data in Temp
  - DO NEXT

Lesen weniger Datensätze

6.2.1 B*-Bäume

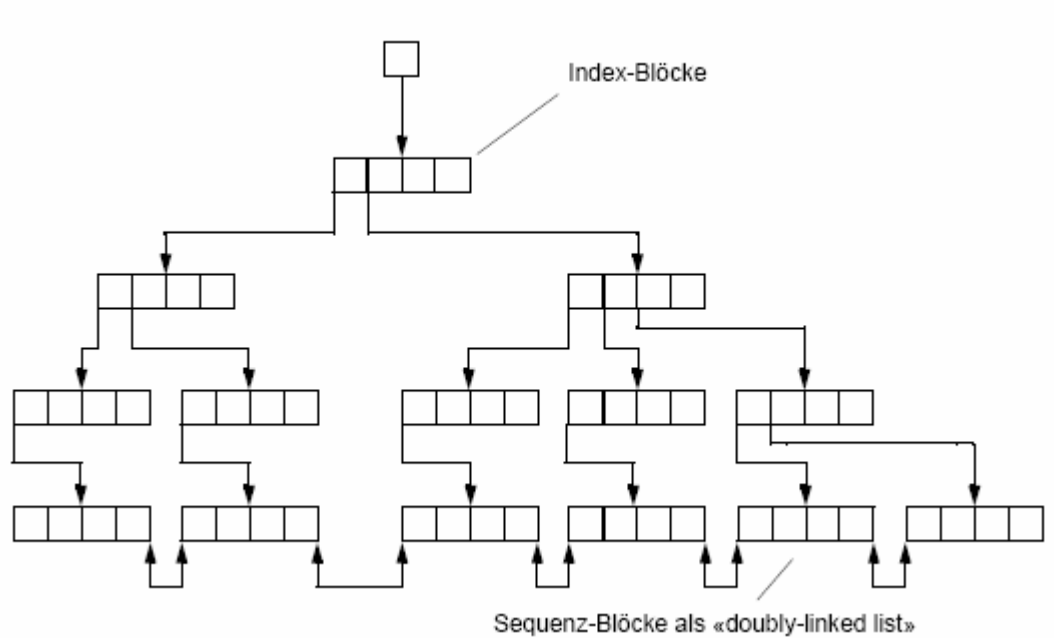
Merkmale von B*-Bäumen

- In ausbalancierten Bäumen können Schlüsselwerte schnell gefunden werden.
- ROWID-Adressen (hexadezimal) bieten direkten Zugriff auf Tabellenzeilen.
- Abfragen, die nur indizierte Spalten referenzieren, können im Index gelöst werden-
- Index-Suche ist eine Alternative zum Full Table Scan; die Index-Suche kann Plattenzugriffe reduzieren.
- B*-Bäume werden automatisch vom Oracle Server verwendet und verwaltet (Füllgrad nicht mindestens 50 %, sondern2/3). Dies vermindert den

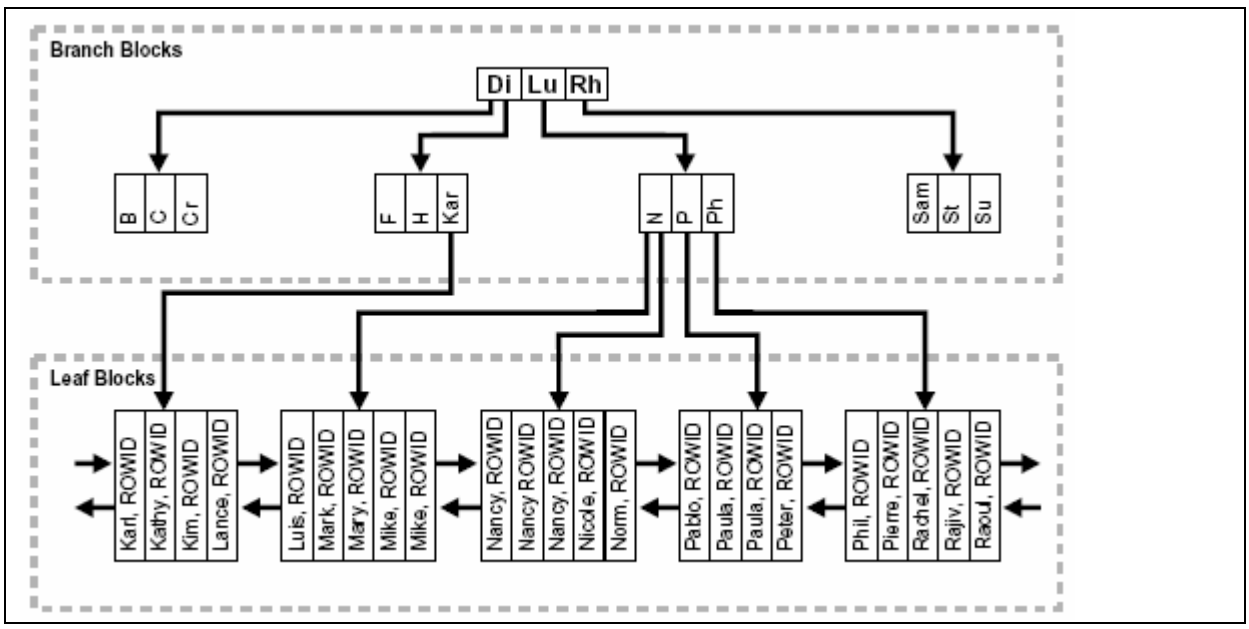
ungünstigsten Fall für den B-Baum, nämlich den Insert mit aufsteigender Reihenfolge (z.B. 1,2,3,4,5,6,7,8,9 etc.)

Grundsätzlichere Aufbau:

B-Trees bestehen aus Index- und Sequenzblöcken. In den Sequenzblöcken werden die ROWID's der Records gespeichert:



Beispiel: B-Baum-Index



Eigenschaften von Index-Segmenten

- Unabhängig von Tabelle oder Cluster

- Jeder Index kann eigene Speicherparameter und Tablespace-Festlegung besitzen
- Indizes sind normalerweise viel kleiner als die indizierte Tabelle
- Können parallel erstellt werden
- Können ohne Protokollierung in der Redo Log-Datei UNRECOVERABLE erstellt werden
- Sollten aus Optimierungsgründen in einem getrennten Tablespace angelegt werden
- Können entweder eindeutig oder nicht eindeutig sein
- Können entweder einfach (eine Spalte) oder zusammengesetzt (verkettet) sein

Beispiel:

```
CREATE INDEX indexname
ON table (spalte)
STORAGE (INITIAL 500 K NEXT 500 K PCTINCREASE 0)
TABLESPACE usr_index;
```

6.2.2 Bitmap-Indizierung

Bietet unter folgenden Voraussetzungen Vorteile gegenüber B*-Bäumen:

- Sehr große Tabellen (Millionen von Zeilen)
- Verschiedenen Werten (z.B. Geschlecht. Familienstand)
- Spalten in Abfragen, die üblicherweise in der WHERE-Klausel vorkommen
- Abfragen mit Bedingungen in der WHERE-Klausel, die von Tausenden von Zeilen erfüllt werden

Konzepte der Bitmap-Indizierung

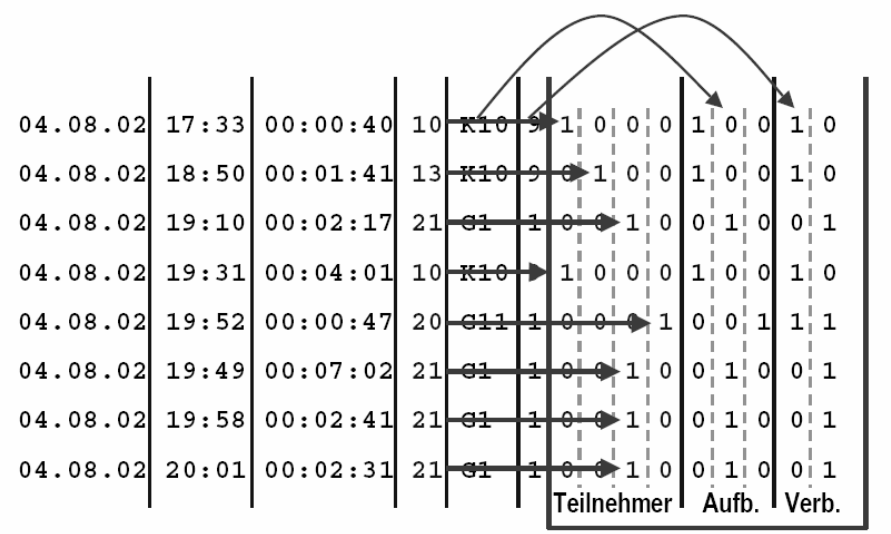
ein Bitmap für jeden Wert. Ein Eintrag für jede Zeile

Spalte = REGION		
Ost	Mitte	West
1	0	0
0	1	0
0	0	1
0	0	1
0	1	0
0	1	0

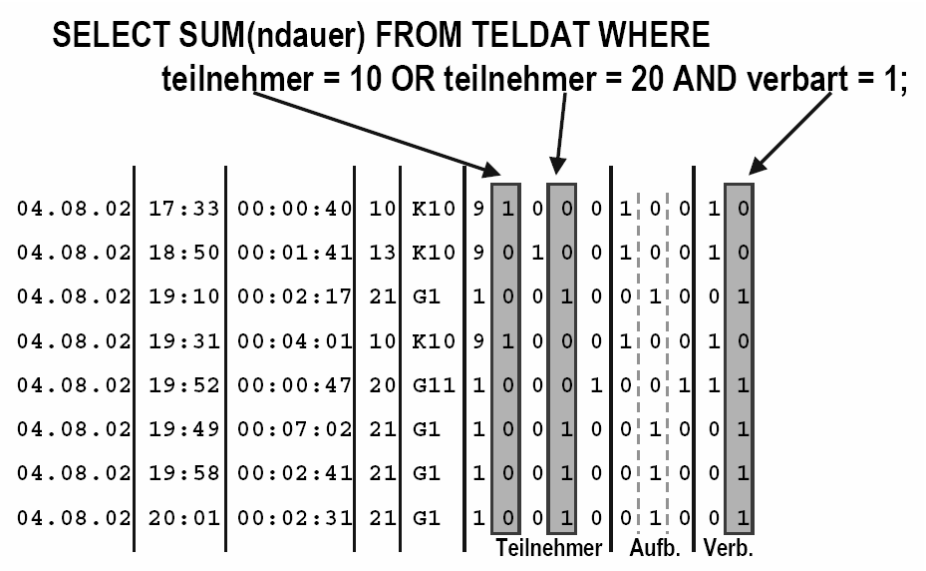
- Effiziente Kombination von Bitmaps durch die Verwendung von logischen Operatoren (AND, OR usw.)
- Erhebliche Verringerung des Speicherplatzes im Vergleich mit anderen Indizierungs- Methoden
- Entscheidende Performance-Verbesserungen bei einigen Abfragen

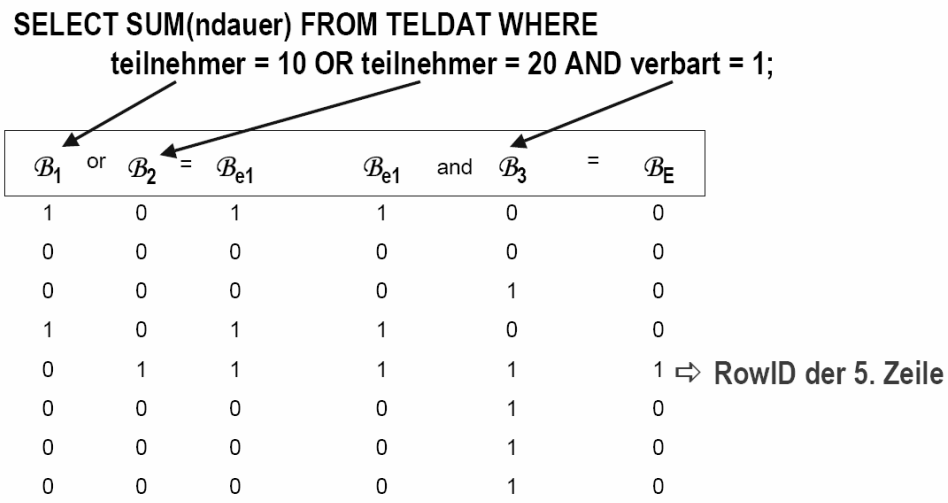
Zugriff auf Bitmap-Index:

Tabelle:



Zugriff:





Wann sollte Bitmap-Indizierung verwendet werden

Bitmap-Indizierung sollte in den folgenden Fällen in Betracht gezogen werden:

Abfragen mit

- Spalten niedriger Kardinalität
- Einer Kombination von mehreren WHERE-Bedingungen
- Bedingungen, die eine große Anzahl von Zeilen betreffen

Als Alternative zu verketteten Indizes bietet sie

- eine effizientere Nutzung des Speicherplatzes
- keine Abhängigkeit von der Spalten-Reihenfolge, wenn nur lesende Aktivitäten oder wenig Aktualisierungen stattfinden

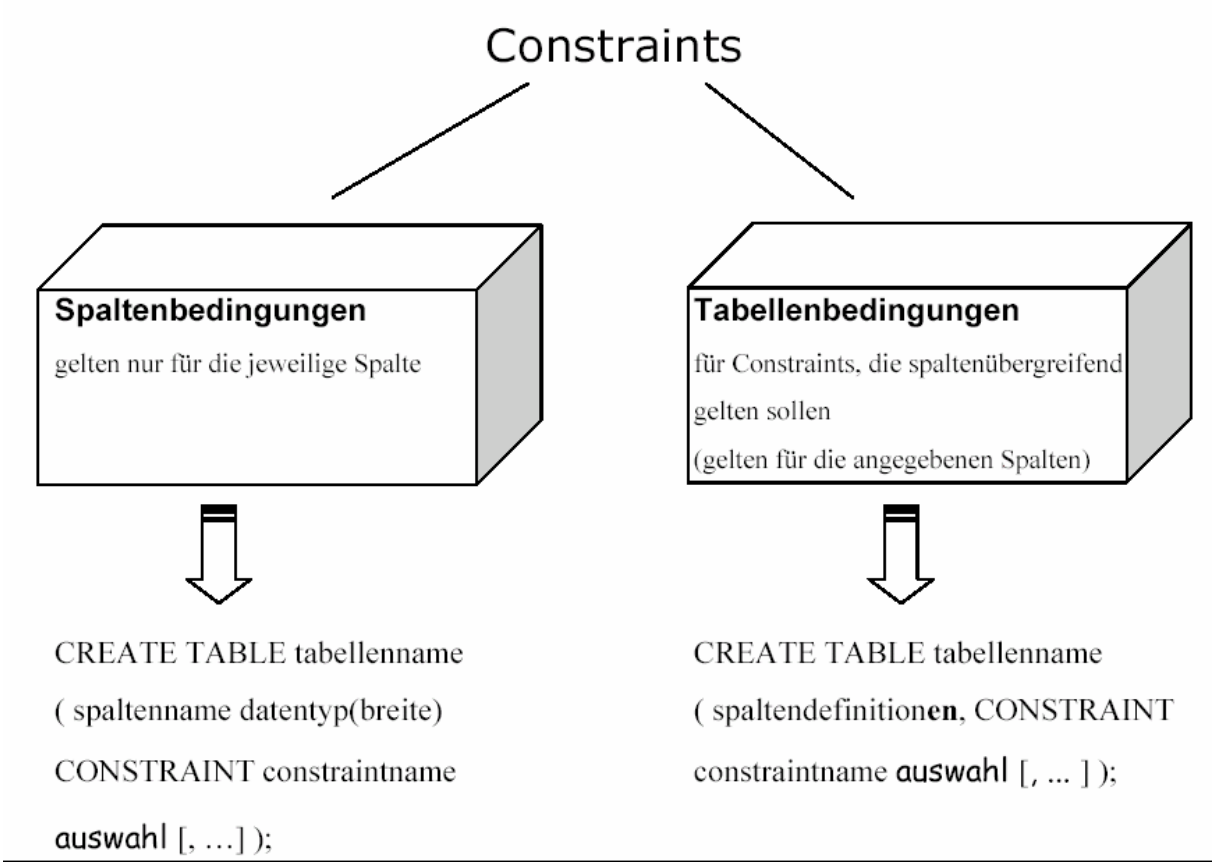
Indizes mit Bitmap-Indizierung sind aufwendig zu aktualisieren.

Einsatz praktisch:

Sehr bedingt bei OLTP
Sehr gut bei OLAP

6.3 Constraints

Die Eigenschaften von Daten einer Tabelle werden über die Integritäts-Constraints festgelegt.



UNIQUE	Eindeutigkeit	UNIQUE (spaltennamen)
PRIMARY KEY	Primär-schlüssel	PRIMARY KEY (spaltennamen)
REFERENCES tabellenname (spaltenname) [ON DELETE CASCADE]	Fremd-schlüssel	FOREIGN KEY (spaltennamen) REFERENCES tabellenname (spaltennamen) [ON DELETE CASCADE]
CHECK (bedingung)	zusätzl. Bedingung	CHECK (bedingung)
NOT NULL	Mußeingabe-feld	—

Definiert werden sie über CREATE TABLE oder ALTER TABLE. Sie können aktiviert, deaktiviert oder gelöscht werden.

Constraint-Verletzungen könne über eine Exception-Tabelle aufgezeichnet werden.

Datenintegrität und Constraints

Datenintegritätstypen:

NULL-Werte
eindeutige Spaltenwerte (Domänenwerte)
eindeutige Zeilenidentifizierung (Primärschlüssel) Achtung! Erzeugt beim einschalten automatisch Index!
referentielle Integrität
komplexe Unternehmensregeln

Definition und Überprüfung über

Integritäts-Constraints
Trigger
Anwendercode

Vorteile deklarativer Integritäts-Constraints

- bessere Performance
- einfach zu deklarieren
- zentrale Regeln
- einfach zu ändern
- sofortiges Benutzerfeedback
- flexibel (einschalten/ausschalten)
- im Data-Dictionary vollständig dokumentiert

Nachteile

Überblick geht rasch verloren

Komplexe Unternehmensregeln könne nur über Trigger realisiert werden.

Beispiel:

```
CREATE TRIGGER increase_but
        BEFORE UPDATE OF salary
        ON s_emp
                FOR EACH ROW WHEN (NEW.salery<OLD.salery
                OR NEW.salery > 1.1 * OLD.salery)
BEGIN
        raise_application_error (-20502,  Kann salery nicht verringern.
        Änderung muss kleiner 10 % sein );
END;
```

Achtung: Es gibt Column-Constraints und Table-Constraints.

Wird auf die Angabe

CONSTRAINT name

verzichtet, wird der Constraint ein interner Name zugeordnet, was dessen Verwaltung im Data Dictionary sehr stark erschwert (SYS_C00n)!

Enthält die Tabelle den Primär- und den Fremdschlüssel, ist sie selbstreferenzierend (z. B. Mitarbeiterspalte, Managerspalte)

Sollen Schlüssel aus einer Tabelle auf die mit Fremdschlüsseln gezeigt wird, gelöscht werden, kann das Löschen nur kaskadierend erfolgen:

Option beim CREATE oder ALTER TABLE: ON DELETE CASCADE: Diese löscht automatisch die Fremdschlüssel-Zeilen, die eine gelöschte Master-Zeile referenzieren.

Check-Constraints

Diese werden verwendet

um zu überprüfen, ob Spaltenwert in einem bestimmten Bereich ist
um zu überprüfen, ob Spaltenwert in einer Liste von Werten vorkommt
um Spaltenwerte mit anderen Spaltenwerten in derselben Tabelle zu vergleichen

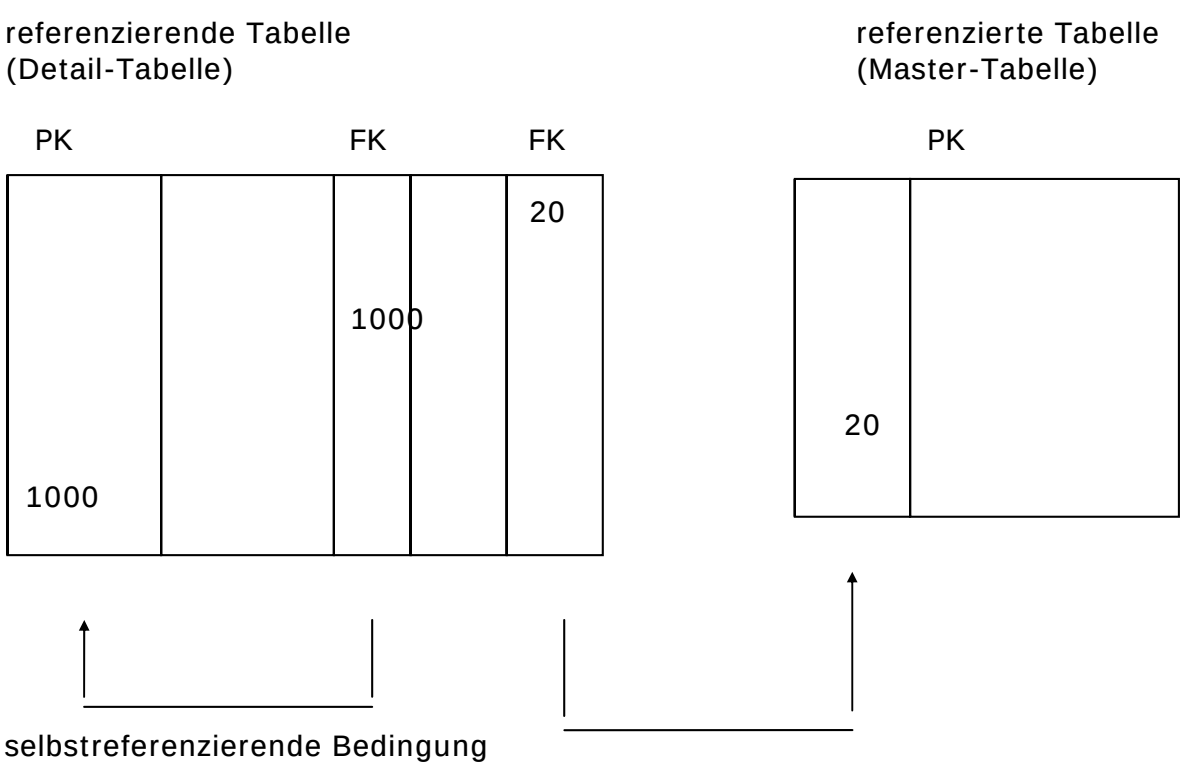
Check-Constraints können

eine oder mehrere Spalten referenzieren
Spalten miteinander oder mit Konstanten vergleichen

Im Gegensatz zu Triggern können Check-Constraints

keine Unterabfragen enthalten
nicht die Pseudospalten CURRVAL und NEXTVAL referenzieren
nicht die SQL-Funktionen SYSDATE, UID, USER und USERENV enthalten, weil sich diese ändern.

Fremdschlüssel-Constraints



Regeln für Sperren betreffend Fremdschlüssel-Constraints:

Kein Index auf den Fremdschlüssel

- Insert in die Vater-Tabelle erfordert keine Sperren auf die Sohn-Tabelle
- Bei einem Delete oder Update in der Vater-Tabelle wir die gesamte Sohn-Tabelle gesperrt. Shared Lock auf die Sohn Tabelle ist erforderlich.
- Ein Insert, Update und Delete in der Sohn-Tabelle erfordert keine Sperren in der Vater-Tabelle.

Index auf dem Fremdschlüssel

Bei einem DELETE oder UPDATE in der Vater-Tabelle werden keine Sperren auf die Sohn-Tabelle gesetzt, wenn es einen Index auf den Fremdschlüssel der Sohn-Tabelle gibt. Daher könne alle DML-Anweisungen in der Vater-Tabelle durchgeführt werden.

Wenn in der Sohn-Tabelle ON DELETE CASCADE definiert ist, kann das Löschen in der Vater-Tabelle ein Löschen in der Sohn-Tabelle zur Folge haben. Sperren und Warten auf Freigabe gelten daher wie bei expliziter Sperre beim Löschen in der Sohn-Tabelle, nachdem das Löschen in der Vater-Tabelle abgeschlossen ist.

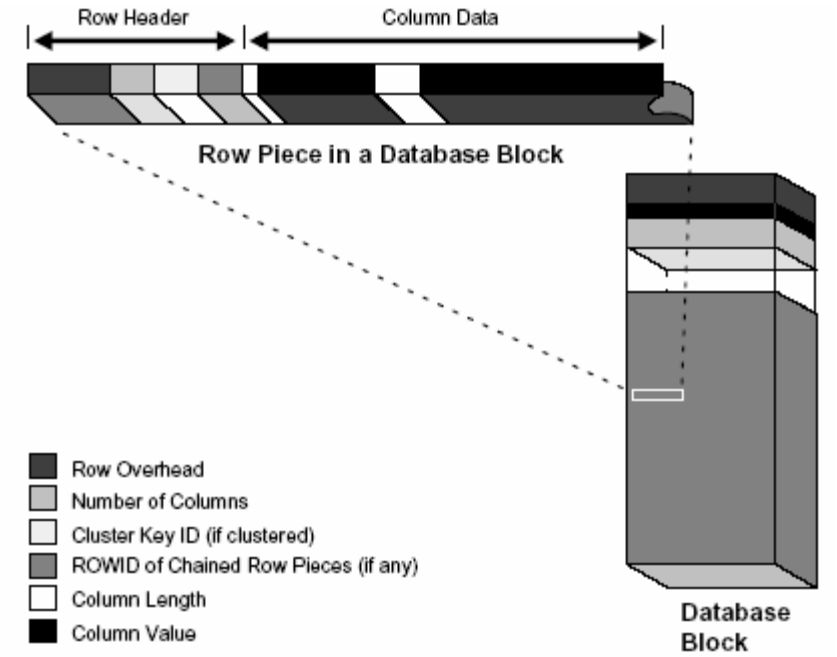
Selects zur Tabellenkontrolle:

ROWID ist als Funktion enthalten

SELECT ROWID, ename FROM emp;

Rowid enthält (aufpassen auf die Oracle-Version hier 9):

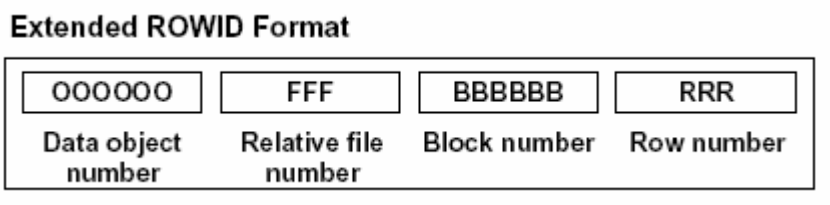
Rowid-Format für Spalten mit weniger als 256 Spalten und nicht verkettete Zeilen. Sie haben eine Länge von 3 Bytes.



The **row header** precedes the data and contains information about:

- Row pieces
- Chaining (for chained row pieces only)
- Columns in the row piece
- Cluster keys (for clustered data only)

Format der Zeilennummer



An Extended ROWID needs 10 bytes of storage on disk and is displayed using 18 characters. It consists of the following components:
• *Data object number* is assigned to each data object, such as table or index when it is created, and it is unique within the database.
• *Relative file number* is unique to each file within a tablespace.
• *Block number* represents the position of the block, containing the row, within the file.
• *Row number* identifies the position of the row directory slot in the block header.
Internally, the data object number needs 32 bits, the relative file number needs 10 bits, block number needs 22 bits, and the row number needs 16 bits, adding up to a total of 80 bits or 10 bytes.
An extended ROWID is displayed using a base-64 encoding scheme, which uses six positions for the data object number, three positions for the relative file number, six positions for the block number, and three positions for the row number. The base-64 encoding scheme uses characters “A-Z,” “a-z,” “0-9,” “+,” and “/”—a total of 64 characters, as in the example below:

Beispiel:

```
SQL> SELECT department_id, rowid FROM hr.departments;
DEPARTMENT_ID ROWID
------------- ------------------
           10 AAABQMAAFAAAAA6AAA
           20 AAABQMAAFAAAAA6AAB
           30 AAABQMAAFAAAAA6AAC
           40 AAABQMAAFAAAAA6AAD
           50 AAABQMAAFAAAAA6AAE
           60 AAABQMAAFAAAAA6AAF
            ...
```

In this example:

• AAABQM is the data object number.
• AAF is the relative file number.
• AAAAA6 is the block number.
• AAA is the row number for the department with ID=10.

DESCRIBE tablename, viewname

In der Tabelle EXCEPTIONS sind alle Ausnahmezustände enthalten.

```
SELECT ROWID, id, dept_id FROM s_emp
WHERE ROWID IN (SELECT row_id FROM exceptions);
```

Wichtige Views

ALL_CONSTRAINTS
ALL_CONS_COLUMNS
USER_CONSTRAINTS
USER_CONS_COLUMNS
DBA_CONSTRAINTS
DBA_CONS_COLUMNS

Zur besseren Ausgabegestaltung kann Breite am Bildschirm angegeben werden:

```
SQL>COL name FORMAT A20

SQL>SELECT name FROM tabelle
```

6.4 Clustersegmente verwalten

Daten-Segmente enthalten die Daten, die in Tabellen eingefügt wurden

Eigenschaften von Daten-Segmenten

Daten-Segment-Typ	Beschreibung
Index-Cluster	Enthält Zeilen aus einer oder mehreren Tabellen, die aufgrund der Werte in einer Spalte der Tabellen zusammen gespeichert werden
Hash-Cluster	Enthält Zeilen einer Tabelle, die nach dem Hash-Algorithmus gespeichert werden

- Cluster definieren eine Speicherstruktur. Jedes Cluster enthält eine oder mehrere Tabellen Definitionen
- Hash-Cluster können mehr als eine Tabelle enthalten, allerdings ist dies selten der Fall.
- Auf Tabellen in Clustern greifen Benutzer genauso zu, wie auf Standalone-Tabellen

Index-Cluster

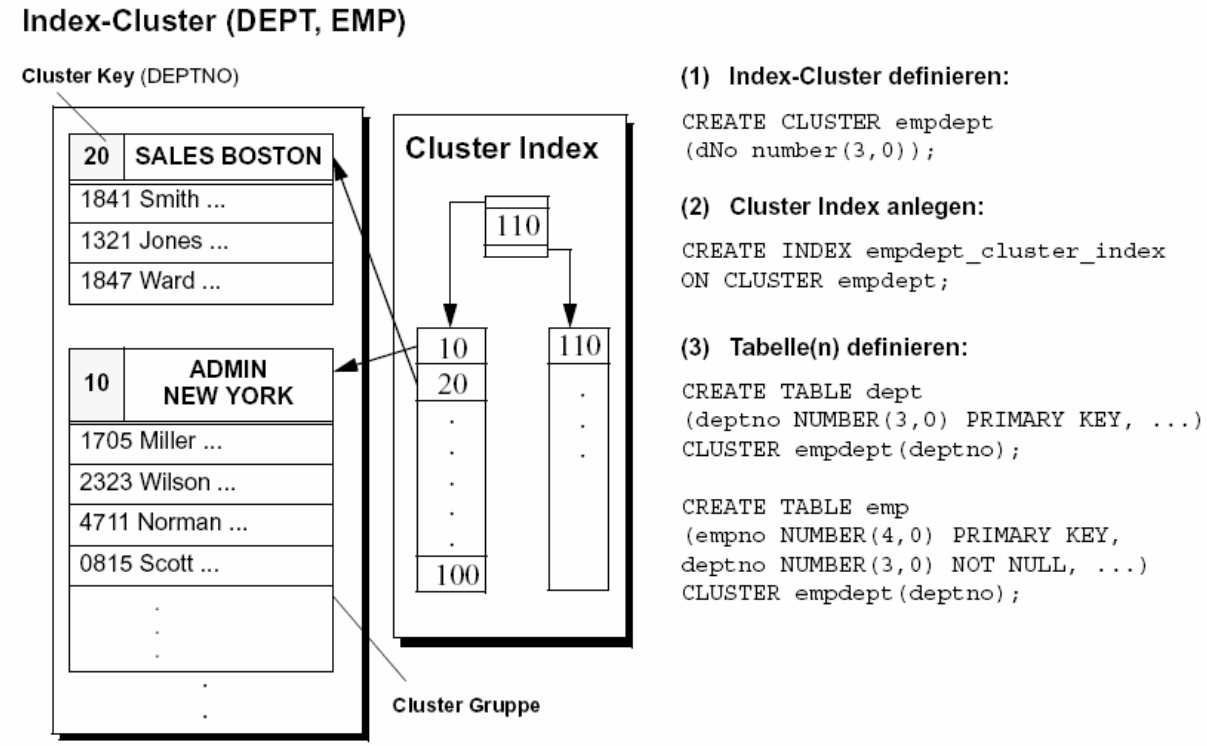
Cluster sind eine optionale Methode zum Speichern von Tabellen, um die E/A-Performance zu verbessern und Speicheraufwand zu reduzieren. Die Spalten, die die Tabellen im Cluster gemeinsam haben, bilden den Cluster-Schlüssel. Der

Cluster-Index ist ein Index, der für die Spalten des Cluster-Schlüssels angelegt wird.

Die Tabellen in einem Cluster

haben gemeinsame Spalten
werden häufig miteinander benutzt
sind in denselben Datenblöcken gespeichert

Beispiel:



Hash-Cluster:

Prinzip: Cluster Key ist das Ergebnis der Anwendung einer Hash-Funktion auf ein oder mehrere Attribut(e). Zu jedem Hash-Wert gibt es ein Hash-Bucket (das in Regel kleiner sein soll als ein Block).

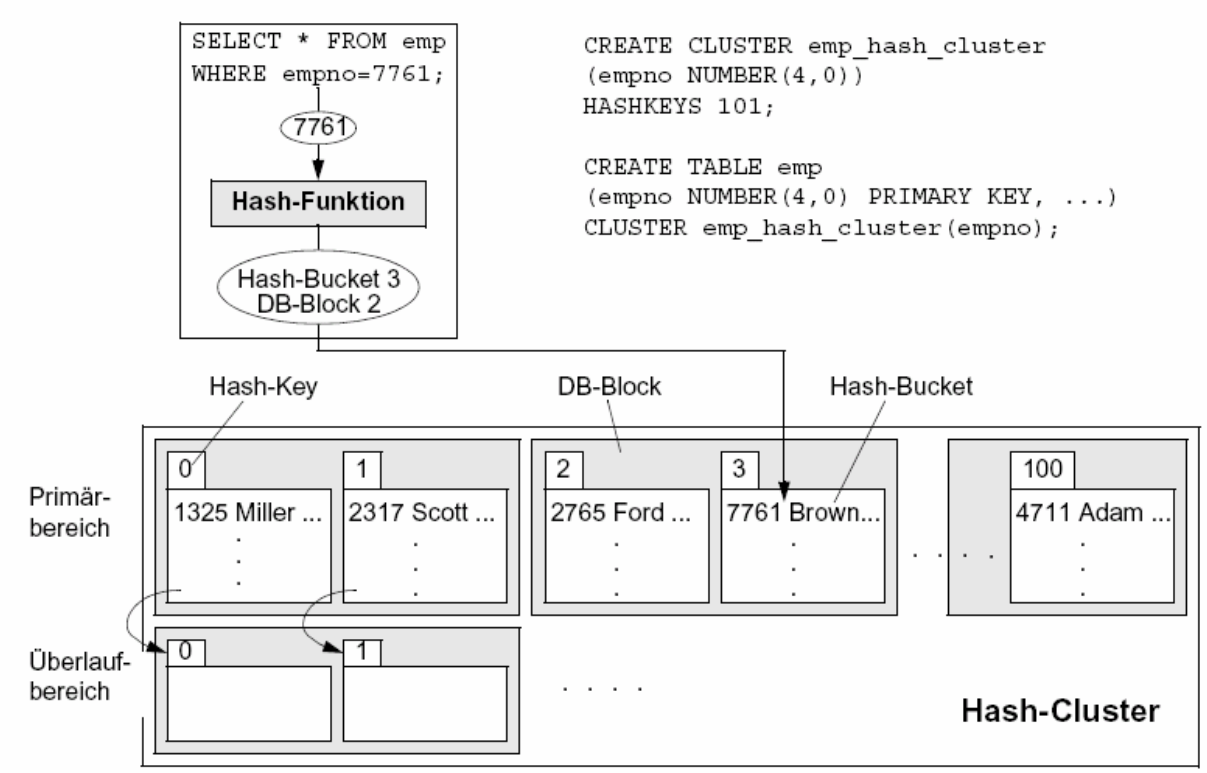
Einsatz: Wird verwendet, um einzelne Tabellen aber auch um mehrere Tabellen gemeinsam zu speichern.

Vorteil: Extrem effizienter Zugriff über die Attributkombination auf die die Hash-Funktion angewendet wird. Es ist keine Suche im Index notwendig!

Hash-Funktion: Als Hash-Funktionen kommen typischerweise Divisions/Restwertverfahren
zur Anwendung. Beispiel für Mitarbeiter-Tabelle: SVNr MOD 101.
Es gibt 101 Restklassen (=Hash-Werte): 0 bis 100. Die Mitarbeiter werden auf 101 Hash-Buckets verteilt.

Unterstützung (Nachteil): Hash Cluster unterstützt nur *point-Queries* effizient!

Beispiel:



Vorteile von Clustern

reduzieren E/A, weil zusammenhängende Zeilen in denselben Datenblöcken gespeichert sind
verbessern Zugriffszeiten für Join-Verknüpfungen von Tabellen, die in Clustern gespeichert sind
reduzieren den Speicherplatz für Indizes besonders dann, wenn der Cluster-Schlüssel viele sich wiederholende Werte hat da der Cluster-Schlüssel nur einmal im Index gespeichert wird

Ein Cluster-Schlüssel kann maximal 16 Spalten enthalten. Das aktuelle Maximum hängt von der Daten-Blockgröße ab, da der Cluster-Schlüssel nicht länger als, grob geschätzt, ein Drittel der Daten-Blockgröße sein kann.

Cluster können die Performance von Abfragen verbessern, dagegen die Performance bei Einfügungen, Veränderungen. Löschungen und FULL TABLE SCANs verschlechtern.

Wenn mehr als eine Tabelle im Cluster abgelegt wird, erfordert das Speichern einer Tabelle im Cluster mehr Datenblöcke als das separate Speichern, da die Datenblöcke in einem Cluster gemeinsam verwendet werden, ansonsten ist der Zusatzaufwand nicht bedeutend. Deshalb verringert sich die Performance bei

DML-Anweisungen Im Allgemeinen sollten häufig veränderte Tabellen nicht im Cluster abgelegt werden.

Anwendungen nutzen Tabellen im Cluster automatisch. Die Existenz von Clustern ist für Benutzer und Anwendungen transparent.

Wenn mehrere Tabellen in einem Cluster abgelegt sind, werden FULL TABLE SCANs langsamer, da die Datensätze aller Tabellen gescannt werden müssen. Ist nur eine Tabelle im Cluster abgelegt, gibt es diesen Nachteil nicht.

7 Benutzer verwalten

Hinweis: Alle Objekte eines Benutzers (Tabellen, Indizes, Views, SQL-Prozeduren,...) werden als **Schema** bezeichnet. Unter dem Begriff Schema wird ebenfalls der logische Rahmen verstanden, in dem ein Benutzer (User) seine Objekte anlegen kann. Jeder Benutzer ist einem Schema zugeordnet, wobei der Name des Schemas gleich dem des Benutzers ist. Dieser Begriff ermöglicht also eine Unterscheidung zwischen verschiedenen Benutzern und ihren Objekten.

Achtung! Zuerst eigene Systemprivilegien prüfen (z.B. DROP USER)

Dies umfasst die Tätigkeiten: anlegen
ändern
löschen
überwachen
Benutzersitzungen beenden

Die Datenbanksicherheit wird durch das Anlegen von Benutzern und zugehörigen Schemata gewährleistet.

Wenn ein Benutzer angelegt wird, wird auch ein **Schema** mit demselben Namen angelegt. Dies umfasst daher:

Cluster
Datenbank-Links
Tabellen
Indizes
Views
Sequences
Synonyme
Stored Procedures
Trigger

Zugriffsrechte für Benutzer mit Einstellungen für die **Sicherheitsdomäne**:

Default-Tablespace: für CREATE TABLE, CREATE INDEX, CREATE CLUSTER

Temporärer Tablespace: Für Sortieren oder Zusammenfassen

Tablespace Quota: Für jeden Tablespace den maximalen Speicherplatz (Quota 0 hat damit keinen Zugriff) in K oder M (z.B. Quota 10M)

Grenzen für Systemresourcen (**Profile**): CPU-Zeit, Anzahl logischer Lesezugriffe, Anzahl gleichzeitiger Sitzungen pro Benutzer, Leerlaufzeit für eine Sitzung.

Angelegt wird Benutzer über

CREATE USER

Geändert werden die Benutzer mit

ALTER USER (Achtung! Hier kommt **Default-Role** dazu)

Es gibt zwei reservierte Benutzer für Datenbank: sys und system (hat weniger Rechte als sys und wird oft gelöscht)

Der Benutzer SYS hat besondere Rechte. Er gilt als Installationsbenutzer. Eine Verbindung zur Datenbank mit der Anmeldung *internal* umgeht die normalen Authorisierungsroutinen und man arbeitet dann als ORACLE-Benutzer SYS. Das Passwort für diese besondere Verbindung zur Datenbank ist als einziges Passwort nicht in der Datenbank abgelegt, sondern in einem eigenen Passwortfile abgespeichert. Der Benutzer SYS darf als einziger Benutzer die Datenbank hoch- und herunterfahren (siehe Instanz und Datenbank).

Der zweite vorkonfigurierte ORACLE-Benutzer ist SYSTEM mit dem Default-Passwort *manager*, der als Datenbankadministrator (DBA) tätig ist. Er darf neue Tablespaces, neue Benutzer usw. einrichten.

Benutzter wird angelegt und es wird ihm mitgeteilt:

Benutzername und Passwort (auch wie Änderung erfolgt)
zugewiesener Tablespace mit entsprechenden Quotas
wie Anmeldung beim Server erfolgt

Benutzer löschen mit:

DROP USER username [CASCADE]

CASCADE: es werden alle zugehörigen Objekte des Schemas mitgelöscht

Überwachen der User über Views

ALL_USERS	alle Benutzer
USER_USERS	aktuelles Benutzer
DBA_TS_QUOTAS	Tablespace Quotas für alle Benutzer
USER_TS_QUOTAS	Tablespace Quotas für aktuellen Benutzer

Abbrechen einer Benutzersitzung

rollt aktuelle Transaktion zurück
alle Tabellen und Zeilensperren werden aufgehoben
alle Ressourcen werden freigegeben

Beispiel:

```
SELECT sid, serial#, username
FROM v$session;
```

SID SERIAL# USERNAME

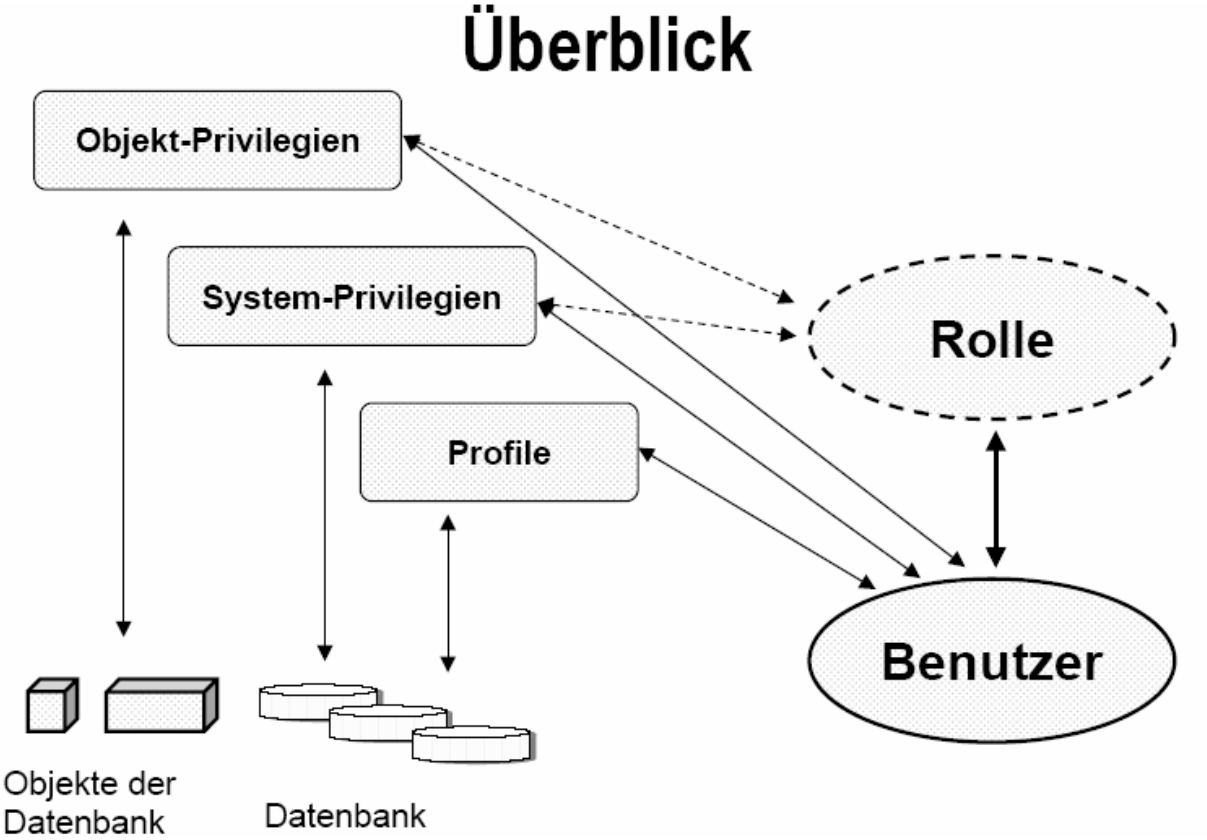
8 103 SCOTT

ALTER SYSTEM
KILL SESSION 8,103 ;

sid = Session ID
serial# = laufende Nummer des Benutzers

8 Ressourcenverwaltung (Benutzerprofile)

Zusammenhang zwischen Rechten und Profilen:



Achtung! Eigene Systemprivilegien überprüfen (z.B. ALTER SESSION)

Die Systemressourcen sind:

CPU-Zeit
E/A-Operationen
Leerlaufzeit
Hauptspeicherplatz privater SQL-Bereich/nur MTS (Multi-Threaded Server)
gleichzeitige Sessions

Achtung: Wenn Grenze auf Sitzungsebene (session-level) erreicht wurde ist nur mehr COMMIT, ROLLBACK oder Abmeldung erlaubt.

Wenn Grenze auf Aufrufebene erreicht wurde, wird Anweisung gestoppt und zurückgerollt.

Zwei Arten für Änderung der Ressourcengrenzen:

Datenbank stoppen und in der init.ora den Parameter **RESOURCE_LIMIT** ändern und Ressourcengrenzenprüfung auf TRUE (Default FALSE) setzen

Im laufenden Betrieb über

ALTER SYSTEM SET RESOURCE_LIMIT=TRUE;

Profil erzeugen mit

CREATE profile name

Profil ändern mit

ALTER profile name

Ressourcen auf Sitzungsebene:

CPU_PER_SESSION	CPU-Zeit in Hunderstelsekunden
SESSION_PER_USER	Anzahl gleichzeitiger Sitzungen pro Benutzername
CONNECT_TIME	verstrichene Verbindungszeit in Minuten (veraltert!)
IDLE_TIME	inaktive Perioden in Minuten
LOGICAL_READS_PER_SESSION	Anzahl Datenblöcke (für physikalisches und logisches Lesen)
	(wegen E/A intensiver Anweisungen)
PPRIVATE_SGA	Privater Speicherplatz in der SGA (nur MTS)

Ressourcen auf Aufrufebene:

CPU_PER_CALL	CPU-Zeit pro Aufruf in 1/100 Sek
LOGICAL_READS_PER_CALL	Anzahl Datenblöcke

Die gesamten Ressourcenkosten werden eingestellt über COMPOSITE_LIMIT

Jede Datenbank enthält das Default-Profil default

Es gilt für alle Benutzer, denen kein eigenes Profil zugewiesen wurde. Das Default-Profil kann auch geändert werden:

ALTER PROFILE default LIMIT .....

Profile zuweisen:

CREATE USER ....
ALTER USER ......

Profile löschen:

DROP PROFILE profile CASCADE

Informationen über Profile

DBA_USERS
USER_RESOURCE_LIMITS
DBA_PROFILES
RESOURCE_COST

9 Datenbankzugriff verwalten

9.1 Datenbankprivilegien - Einführung

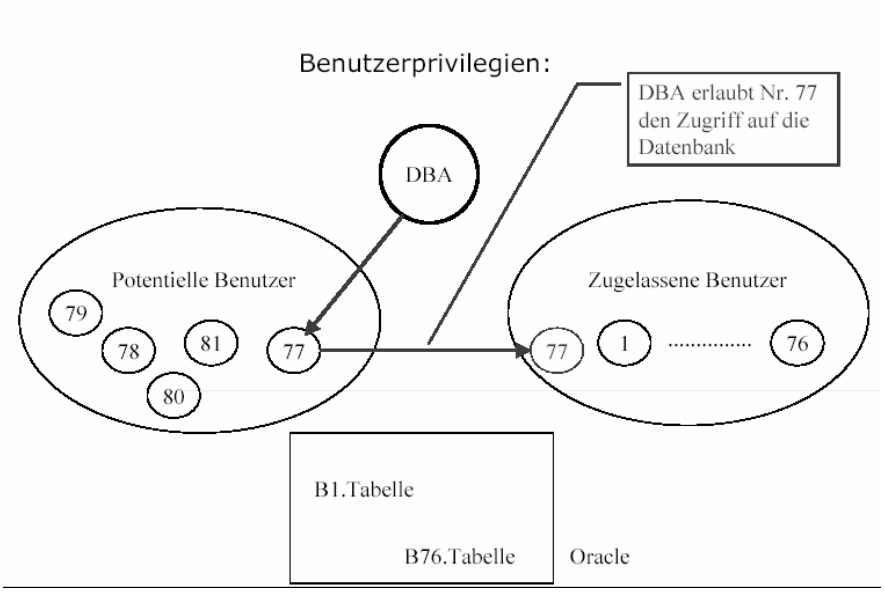
Ein Privileg ist das Recht, einen bestimmten Typ von SQL-Anweisungen auszuführen, oder das Recht, auf ein Objekt eines anderen Benutzers zuzugreifen.

Steuerung der Privilegien:

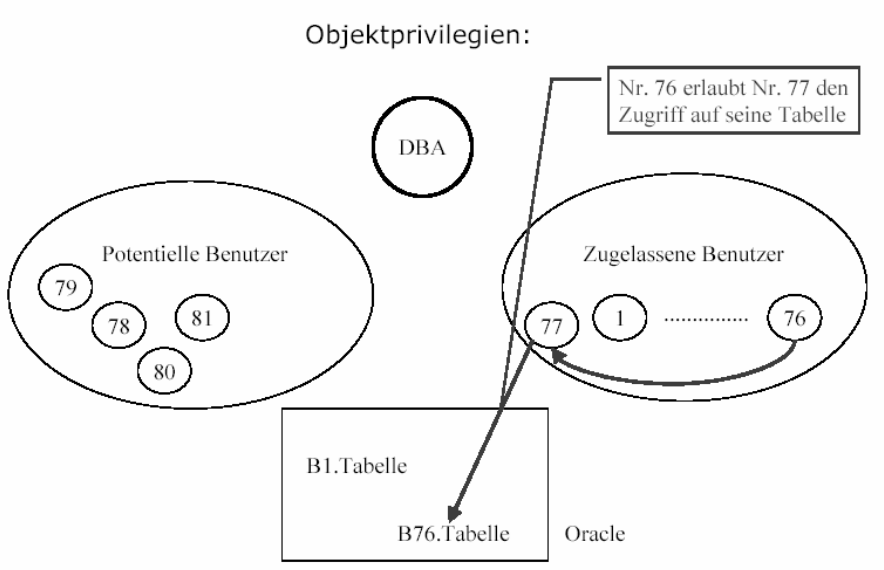
Benutzern Rechte für bestimmte Operationen geben
Zugriff und Änderung von Daten erlauben oder einschränken
Durchführen von Systemfunktionen und Änderung der Datenbankstruktur erlauben oder einschränken
einzelnen Benutzern und Rollen Privilegien erteilen
allen Benutzern Privilegien (PUBLIC) vergeben

Zwei Arten von Privilegien

Systemprivilegien: Es erlaubt einen Benutzer, eine bestimmte Datenbankoperation oder eine bestimmte Klasse von Datenbankoperationen auszuführen.



Objektprivilegien: Es erlaubt einen Benutzer, eine Aktion für eine bestimmte Tabelle, View, Prozedur, Funktion oder Package auszuführen.



Privilegien können durch Rollen vergeben und gesteuert werden. Eine Rolle besteht aus einer benannten Gruppe von Privilegien.

Eigenschaften von Rollen:

reduzierte Vergabe von Privilegien
dynamische Privilegienverwaltung
selektive Verfügbarkeit von Privilegien
abgestimmt auf die Anwendung

Systemprivilegien:

CREATE SESSION
CREATE TABLE
SELECT ANY TABLE

Typen von Systemprivilegien:

Privilegien im eigenen Schema:	Tabellen anlegen Sequences anlegen
Privilegien auf allen Objekten eines Typs: View	Tabellen auf jedem Schema anlegen in jedem Schema Zeilen in jeder Tabelle und aktualisieren
Privilegien für das System oder einen Benutzer:	Benutzer anlegen Sitzung öffnen (anmelden bei einer Datenbank)

GRANT role, priv TO user, role, PUBLIC WITH ADMIN OPTION

PUBLIC Systemprivileg oder Rolle an alle Benutzer
WITH ADMIN OPTION erlaubt Berechtigungsempfänger die Rolle oder Privileg an andere Benutzer weiterzugeben, zu ändern oder zu löschen

Achtung die Rücknahme einer ADMIN OPTION kaskadiert nicht (ist nicht hierarchisch)

SYSDBA und SYSOPER erfordern eine Sonderbehandlung!

Wichtige View:

DBA_SYS_PRIVS

Entziehen eines Privilegs mit:

REVOKE ....

Achtung: Beim entziehen können Systemprivilegien, die weitergegeben wurden, nicht kakadiert entzogen werden!!!

Objektprivilegien:

Object Privileg	Table	View	Sequence	Procedure Function	Snapshot
ALTER	X		X		
EXECUTE				X	
DELETE	X	X			
INDEX	X				
INSERT	X	X			
REFERENCES	X				
SELECT	X	X	X		X
UPDATE	X	X			

REFERENCES: FOREIGN KEY Integritäts-Constraints, die über CREATE TABLE oder ALTER TABLE erzeugt wurden

Die Zuweisung erfolgt über:

GRANT .....

Entzogen werden Privilegien über:

REVOKE .....

Hinweis: References werden mit CASCADE CONSTRAINTS wieder entzogen.

Achtung: Werden Privilegien WITH GRANT OPTION weitergegeben, so werden sie bei auch allen weiteren wieder entzogen.

Diese Privilegien können nicht an Rollen vergeben werden.

Wichtige Views:

DBA_TAB_PRIVS
DBA_COL_PRIVS
USER_TAB_PRIVS
USER_TAB_PRIVS_MADE
USER_TAB_PRIVS_RECD
USER_COL_PRIVS_MADE
USER_COL_PRIVS_RECD
ALL_TAB_PRIVS
ALL_TAB_PRIVS_MADE
ALL_TAB_PRIVS_RECD
TABLE_PRIVILEGS
ALL_COL_PRIVS
ALL_COL_PRIVS_MADE
ALL_COL_PRIVS_RECD
COLUMN_PRIVILEGES

9.2 Rechte vergeben und entziehen

(rekursive) Vergabe von System- und Objektprivilegien mit GRANT

Vergabe von Systemprivilegien

```
GRANT {system_priv | role}, ... TO {user, role, PUBLIC}, ...
  WITH ADMIN OPTION
```

Beispiele:

```
GRANT CREATE TABLE, ALTER SESSION
  TO scott

GRANT manager TO willi
  WITH ADMIN OPTION

GRANT dba TO chef
```

Vergabe von Rechten an Datenbank-Objekten

```
GRANT {object-privilege | ALL}, ... ON [schema.]object
  TO {user, role, PUBLIC}, ...
  [WITH GRANT OPTION]
```

Objekt-Privilegien können sein:

ALTER	DELETE	INDEX	INSERT
REFERENCES	SELECT	UPDATE	

Beispiele:

```
GRANT SELECT, UPDATE
  ON unwichtig TO PUBLIC

GRANT SELECT ON chef.sequenz1
  TO prinz

GRANT REFERENCES (angnr), UPDATE (angnr, gehalt, provision)
  ON mueller.ang TO meier
```

Entzug von System- und Objektprivilegien mit **REVOKE**

Entzug von Systemprivilegien:

```
REVOKE {system_priv | role}, ... FROM {user, role, PUBLIC},
```

Beispiel:

```
REVOKE CREATE TABLESPACE
  FROM verwalter
```

Entzug von Rechten an Datenbank-Objekten:

```
REVOKE {object-privilege | ALL}, ...
  ON [schema.]object
  FROM {user, role, PUBLIC}, ...
  [CASCADE CONSTRAINTS]
```

Beispiel:

```
REVOKE DELETE ON bonus
  FROM meier
```

9.3 Rollen - Benutzerklassen und Zugriffskontrolle

Ähnlich wie das UNIX-System kennen auch Datenbanksysteme wie ORACLE mehrere Benutzerklassen mit unterschiedlichen Rechten und haben Betriebssystem-ähnliche Funktion. Die klassische Dreiteilung (mit absteigender Berechtigung) kennt nur:

den Datenbankadministrator (DBA),
den Benutzer mit RESOURCE-Recht und
den Benutzer, der lediglich das CONNECT-Recht hat.

Die oben genannten Begriffe sind ausschließlich innerhalb der DB wichtig. Aus Sicht des Betriebssystems sind noch einige weitere Voraussetzungen und Berechtigungen notwendig, die vom jeweils verwendeten Datenbanksystem abhängen.

Wird eine neue Datenbank erzeugt, sind die Rechte in einem UNIX-System standardmäßig auf oracle / dba , unter NT auf internal/oracle gesetzt. Jede neue Datenbank erhält automatisch die Datenbankadministratoren SYSTEM und SYS (Vergleiche die jeweilige Installationsanweisung!).

Allgemeines zu Zugriffsrechten

Andere Benutzer haben erst dann den Zugriff auf die Datenbank, wenn ihnen von einem DBA mindestens die CONNECT-Berechtigung erteilt wird (In neueren ORACLE-Versionen (ab Oracle 8) ist bei interaktiver Nutzung zusätzlich die CREATE SESSION-Erlaubnis notwendig.).
Sollen diese Benutzer auch Tabellen und Indizes erzeugen oder löschen können, so muss ihnen ein DBA oder der Eigentümer der Datenbank zusätzlich die RESOURCE-Berechtigung gewähren.
Der DBA-Status kann ebenfalls an einen anderen Benutzer vergeben werden. Dadurch erhält dieser (fast) dieselben Rechte wie der ursprüngliche DBA, einschließlich der Möglichkeit, CONNECT, RESOURCE und DBA-Berechtigungen zu vergeben und zu entziehen.
Ein Benutzer mit DBA-Berechtigung hat sowohl die RESOURCE- als auch die CONNECT-Berechtigung, ein Benutzer mit RESOURCE-Berechtigung verfügt automatisch über die CONNECT-Erlaubnis.
Für das Starten und Stoppen der Datenbank sind zusätzlich zur DBA-Funktion u. U. weitere Voraussetzungen zu erfüllen, z. B. eine bestimmte Gruppenzugehörigkeit (in UNIX-Systemen s. /etc/group).
Zusätzlich zu den genannten Berechtigungen auf Datenbank-Ebene (Systemprivilegien) existieren noch verschiedene Berechtigungen auf Schemaobjekten (Objektprivilegien).
Zunächst kann jeder Objektprivilegien vergeben, der das jeweilige Objekt angelegt hat.
Mit dem Zusatz WITH GRANT/ADMIN OPTION kann die Berechtigung zur Weitergabe des jeweiligen Rechts ebenfalls vergeben werden.

Der Begriff der Rolle

Die oben beschriebene Dreiteilung der Datenbanknutzer wird seit der Version 7 von ORACLE durch den Begriff der "Rolle" (SQL3-kompatibel) erweitert. Unter einer Rolle kann man sich eine Sammlung von Einzelrechten vorstellen; es ist mit diesem Instrument z. B. möglich, Datenbankbenutzer einzurichten, die zwischen RESOURCE- und DBA-Niveau liegen, indem man ihnen eine Rolle Manager zuweist. Um den Administrationsaufwand gering zu halten, können Rollen wiederum anderen Rollen zugewiesen und somit Rechte rekursiv vergeben werden.

Rollen, die standardmäßig von ORACLE geführt werden, sind u. a.:

DBA RESOURCE CONNECT

EXP_FULL_DATABASE IMP_FULL_DATABASE

Die alte Dreiteilung ordnet sich also nahtlos in die neue Rollensystematik ein. Benutzerdefinierte Rollen können mit den nachstehenden Anweisungen bearbeitet werden.

CREATE ROLE erstellt eine Rolle; dieser können einzelne Rechte zugewiesen oder entzogen werden

CREATE ROLE role

[NOT IDENTIFIED |{IDENTIFIED BY password|EXTERNALLY}]

Beispiel:

CREATE ROLE sachbearbeiter IDENTIFIED BY cashflow

ALTER ROLE ändert die Autorisation, die für den Zugriff auf eine Rolle nötig ist

ALTER ROLE role
NOT IDENTIFIED | {IDENTIFIED BY password | EXTERNALLY}

DROP ROLE entfernt eine Rolle aus der Datenbank

DROP ROLE role

SET ROLE Aktivieren/Deaktivieren von Rollen für die aktuelle Sitzung

SET ROLE [role [IDENTIFIED BY password]], ... |
[ALL [EXCEPT role, ...]] | [NONE]

Beispiel:

SET ROLE sachbearbeiter IDENTIFIED BY cashflow

10 Datenbankaudit

Im Bereich Netzwerksicherheit entspricht Auditing dem Aufzeichnen von sicherheitsrelevanten Systemereignissen (z.B. mehrfach fehlgeschlagener Login-Versuch) in Logdateien, deren elementare Informationseinheiten als Audit Records bezeichnet werden.

Mit dem folgenden Kommando lassen sich alle Aktivitäten, die mittels SQL durchgeführt werden, protokollieren. Dies ist anweisungs- oder objektbezogen möglich.
Befehle:
Wählen eines Auditing für spezifizierte SQL-Anweisungen
Wählen eines Auditing für Operationen auf Schema-Objekten
Für SQL-Statements:
AUDIT statement_opt | system_priv, ...
[BY USER]

[BY SESSION | ACCESS]
[WHENEVER [NOT] SUCCESSFUL]

Beispiele:

AUDIT ROLE

AUDIT ROLE WHENEVER SUCCESSFUL

AUDIT SELECT TABLE, UPDATE TABLE BY scott

AUDIT DELETE ANY TABLE

Für Schema-Objekte:
AUDIT object_opt, ... ON [schema.]object | DEFAULT
[BY SESSION | ACCESS]
[WHENEVER [NOT] SUCCESSFUL]

Beispiel:

```
AUDIT SELECT ON buchhaltung.ang
  WHENEVER NOT SUCCESSFUL
```

NOAUDIT
Deaktiviert das Auditing, indem es die Wirkung eines vorhergehenden AUDIT-Kommandos teilweise oder ganz rückgängig macht

Für SQL-Statements:

```
NOAUDIT statement_opt | system_priv, ...
  [BY user, ...]
  [WHENEVER [NOT] SUCCESSFUL]
```

Beispiele:

```
NOAUDIT ROLE

NOAUDIT SELECT TABLE BY scott
```

Für Schema-Objekte:

```
NOAUDIT object_opt, ... ON [schema.]object
  [WHENEVER [NOT] SUCCESSFUL]
```

Beispiel

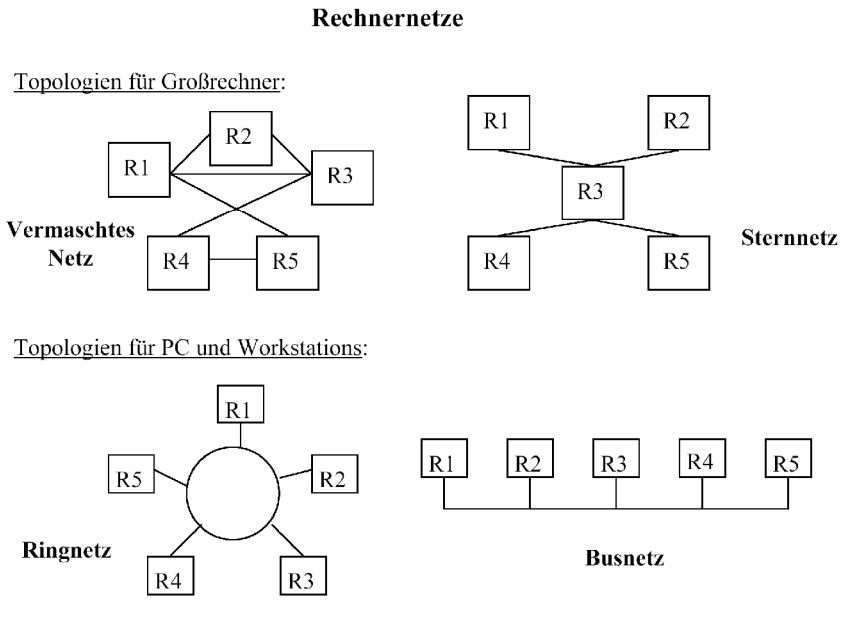
```
NOAUDIT SELECT ON buchhaltung.ang
```

11 Serverkonfigurationen

11.1 Allgemeines zur Serverkonfiguration

Hinweis: Hier werden auch allgemeine Client-Serverprobleme mit berücksichtigt

Netze im Vergleich:



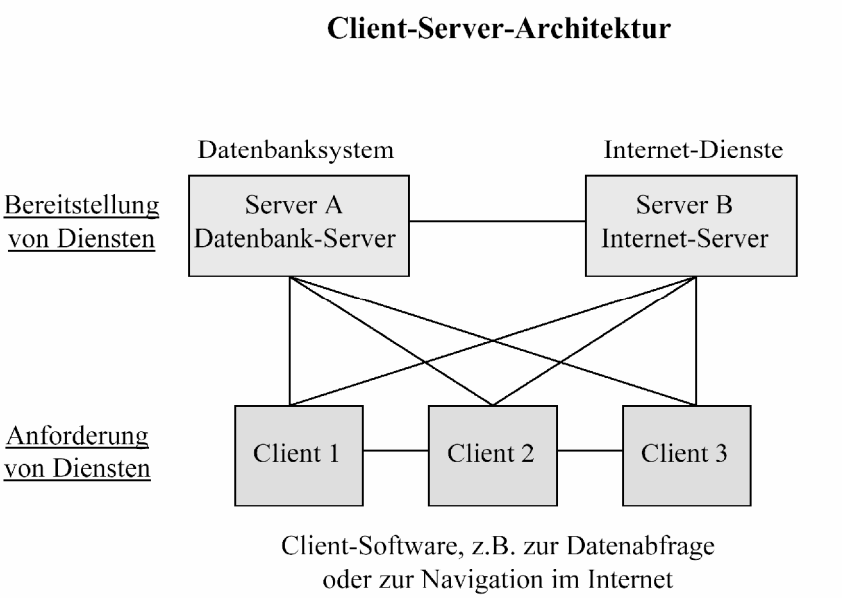
Ziele von Rechnernetzen:

Lastverbund
Datenverbund
Programmverbund
Kommunikationsverbund
Geräteverbund
Sicherheitsverbund

Client-Server-Architekturen

Sehr enger Zusammenhang, aber zu unterscheidenden von Netzen

Zusammenarbeit von Programmen , die auf mehreren Rechnern verteilt sein können



Client/Server-Interaktionsmodell

Client = Dienst-Nachfrager:

Programm, das Aufträge formuliert und an Server verschickt
kann Aufträge auch an mehrere Server vergeben

Server = Dienst-Anbieter:

erhält Aufträge vom Client
bearbeitet die Aufträge und schickt Ergebnisse an den Client zurück
kann seine Dienste mehreren Clients zur Verfügung stellen
kann selbst zum Client werden, indem er im Verlaufe seiner Bearbeitung seinerseits Aufträge vergibt

Monolithische Systeme:

Funktionalität und Datenverwaltung bilden eine untrennbare Einheit
Integrationsfeindliche Systeme

Client-Server-Konzept:

Einteilung von Softwaresystemen in Anbieter (Server) und Nachfrager (Clients) von Software-Dienstleistungen

Übertragung auf Hardware-Systeme:

Rechner, die (Software-)Dienste anderen Rechnern (Clients) anbieten, werden Server genannt

Alternativen:

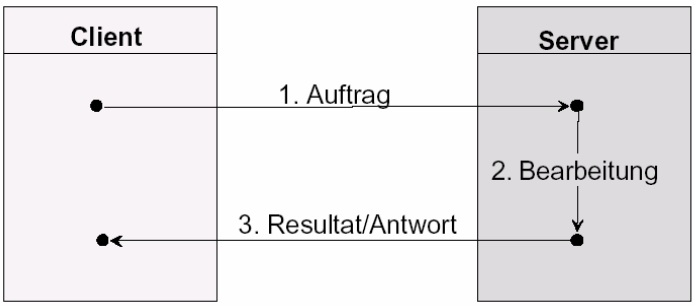
Peer-to-peer

Broadcasting

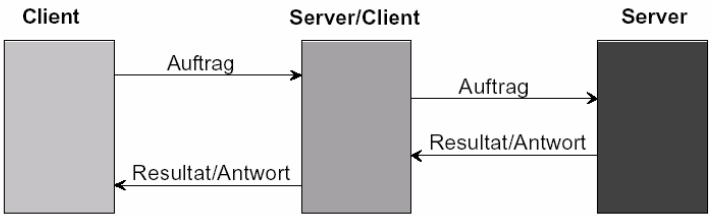
C/S - Arbeiten mit verteilten Systemen

- **Grundschema der Kooperation**
  - Die Initiative zu einer Interaktion geht stets vom Klienten (*Client*) aus.
  - Der Client formuliert Aufträge und schickt diese zur Ausführung an einen Dienstanbieter (*Server*).
  - Für jeden Server ist festgelegt bzw. bekannt, welche Arten von Diensten (*Services*) er anbietet.

Interaktionen im C/S-Modell



Verschiedene Rollen bei Ausführung eines Auftrages



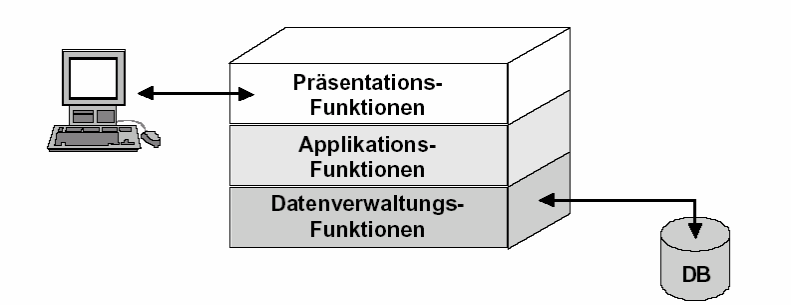
Komponenten einer Applikation

- **Komponenten einer Applikation**
  - ***Präsentationsfunktion***

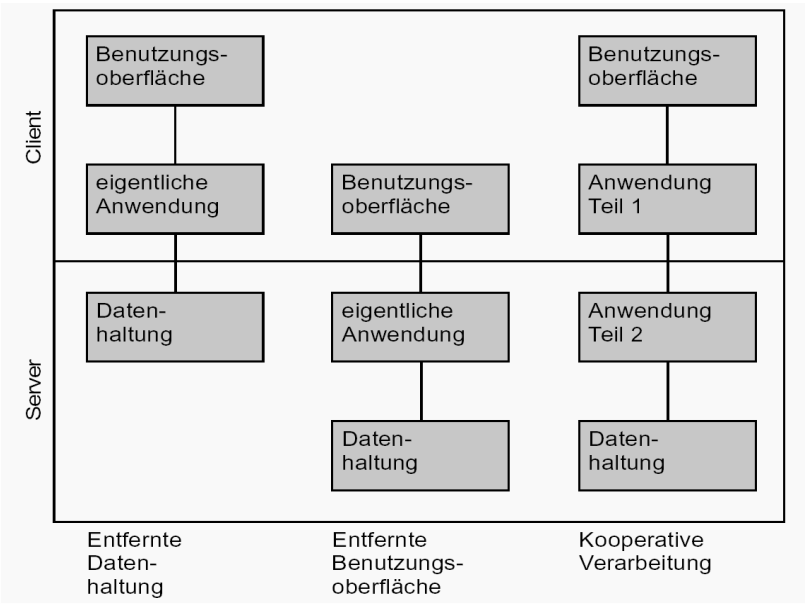
    Gesamte Handhabung der Ein-/Ausgabe (Tastatur, Maus, ..., Bildschirm, Drucker, ...)
  - ***Applikationsfunktion***

    Eigentliche anwendungsspezifische Programm- und Ablauflogik; benutzt Präsentations- und Datenverwaltungsfunktionen (z.B. mittels embedded SQL) zur Realisierung
  - ***Datenverwaltungsfunktion***

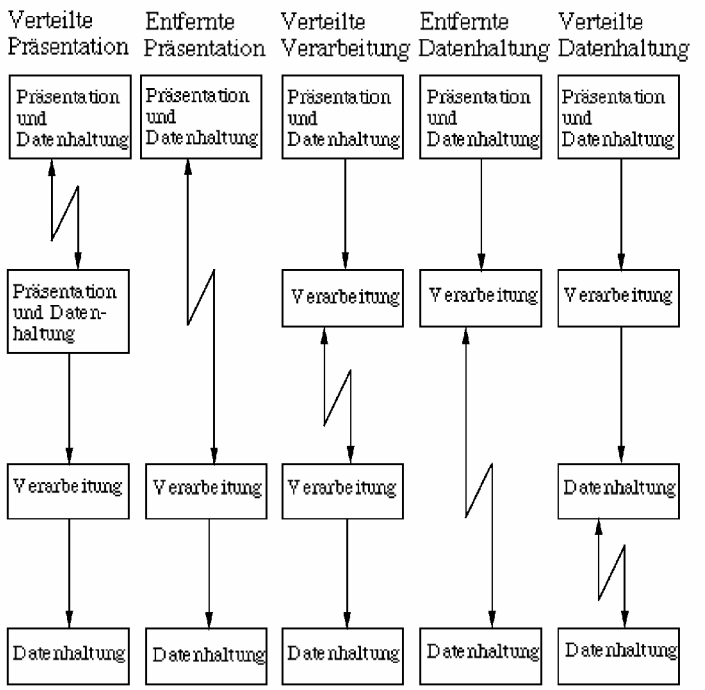
    Realisieren den physischen Zugriff auf die Daten mittels Datei- oder DB-Schnittstelle.



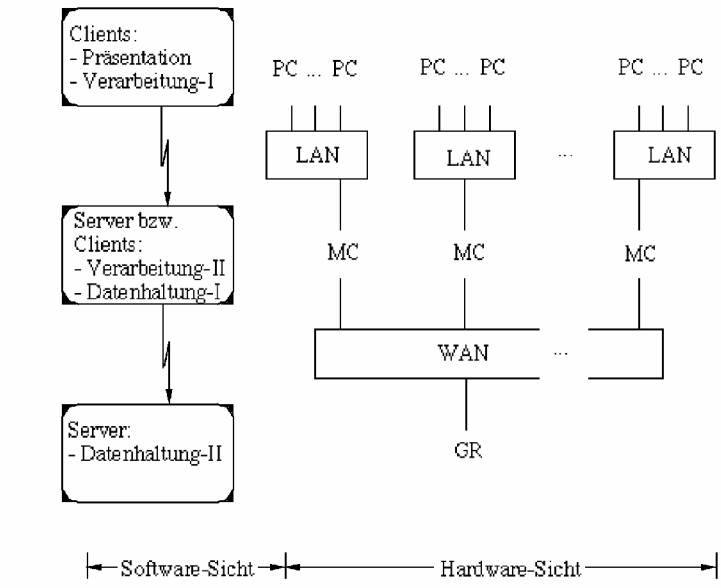
Verteilungsalternative für Client / Server-Konzept



Anders dargestellt:



Beispiel:



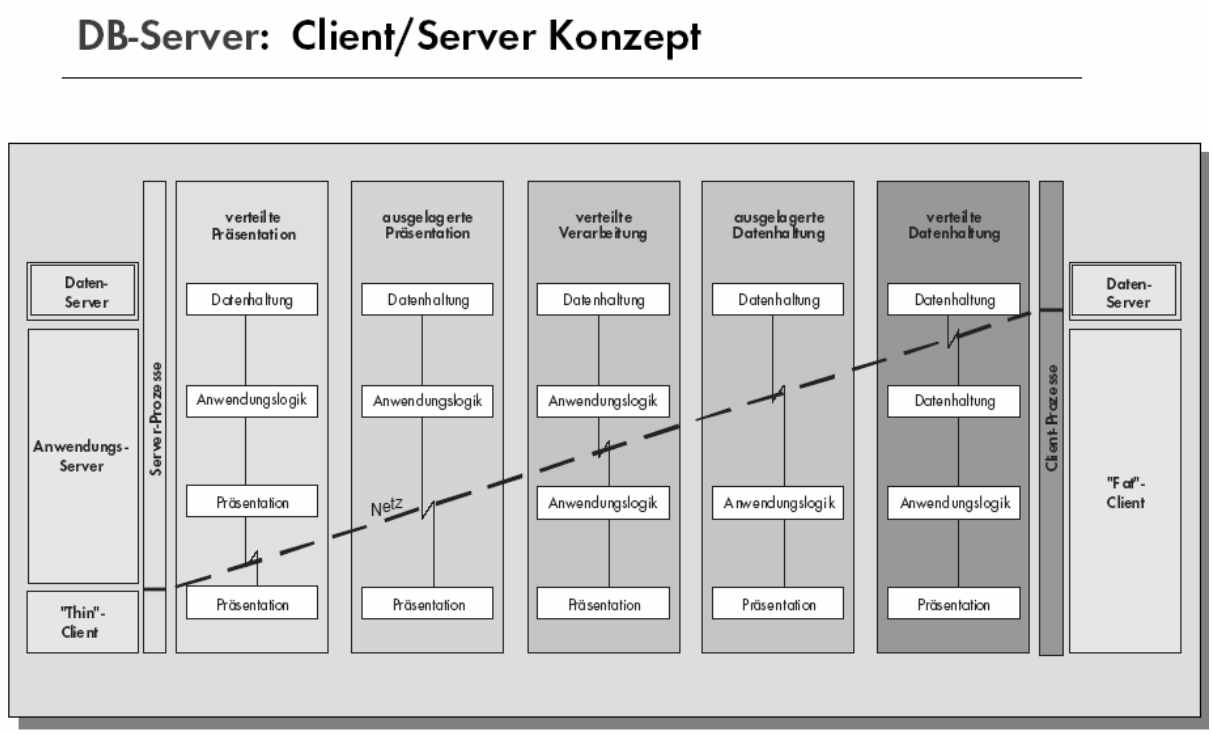
Minicomputer (MC)

lokales Netz (LAN, local area network)

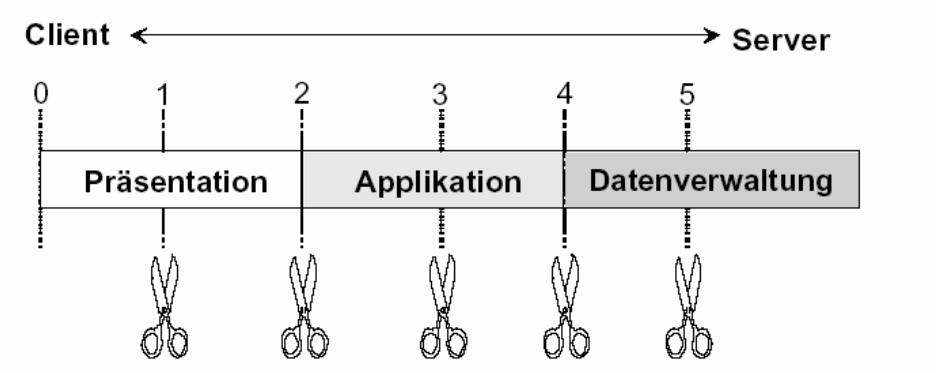
Großrechner (GR)

Weitverkehrsnetz (WAN, wide area network)

Thin / Fat Client



Mögliche Funktionsverteilung zwischen C/S



Schnitt 0: (= keine Zerlegung) Typische *Zentralrechneranwendung*: Vom Datenzugriff über Applikationslogik bis zum Bildschirmaufbau wird alles vom Zentralsystem erledigt.

Schnitt 1: *Verteilte Präsentation*. Die Präsentationsfunktion wird hierbei in eine Server- und eine Client-Komponente aufgespalten

Beispiel: X-Windows (Entwicklung des MIT 1984)

Schnitt 2: *Entfernte Präsentation*. Vom Grundgedanken her traditionelle Terminal-Host-Konfiguration. Die Applikation steuert, im Gegensatz zur verteilten Präsentation, den Bildschirm durch entsprechende Kommandosequenzen unmittelbar selbst.

Ein Terminalemulationsprogramm auf einem PC, das mit einem Hostsystem verbunden ist, wäre ein Beispiel für eine solche C/S-Variante.

Schnitt 3: *Verteilte Applikationsfunktion.* Ein Teil der Applikation auf dem Client-, ein anderer auf dem Serversystem ausgeführt

- Merkmal: Ein Teil der Applikation wird auf dem Client- ein anderer auf dem Server-System ausgeführt.
- Problemstellung in der Regel: Verteilung so wählen, daß Kommunikationsaufwand möglichst gering wird

Schnitt 4 C/S-Variante mit *entferntem Datenbankzugriff* (*remote database access; RDA*), der über eine semantisch hohe Schnittstelle (wie z. B. SQL) realisiert wird

Schnitt 5 C/S-Variante, bei welcher der Server auf einen reinen *Dateiserver (file server)* bzw. *Pageserver* reduziert wird. Die Umsetzung logische Satzadresse (z.B. über TID) in physische Adresse wird clientseitig durchgeführt.

Beispiel zu Schnitt 3:

Gegeben sei ein Informationssystem, das Bestellungen verwaltet.

- *Relation Lieferanten*
  - *AnzBestGesamt* = die Anzahl der aktuell noch offenen Bestellungen bei diesem Lieferanten
  - *BestSummeGesamt* = (ggf. restlicher) Gesamtwert dieser Bestellungen an.
- *Relation Bestellungen*
  - *AnzPosten* = aktuelle Anzahl der (ggf. restlichen) Bestellpositionen dieser Bestellung
  - *BestSumme* = (ggf. restlicher) Gesamtwert dieser Positionen (= Summe über alle BestWert-Attribute einer Bestellung).
- *Relation BestellPositionen*

  = die einzelnen Positionen der Bestellungen

Lieferanten	LiefNr	LiefName	...	AnzBestGesamt	BestSummeGesamt

Bestellungen	BestNr	LiefNr	...	AnzPosten	BestSumme

BestellPositionen	BestNr	PosNr	ArtNr	...	BestWert

Datenbankzugriffe für die Implementierung der Teilfunktion "Verbuchung Warenzugang":

1. SELECT-Zugriff auf *Lieferant*: über LiefNr oder LiefName.
2. SELECT-Zugriff auf *Bestellungen*: über LiefNr.
3. SELECT-Zugriff auf *BestellPositionen*: über BestNr.
4. DELETE-Zugriff auf *BestellPositionen*: ein Tupel wird gelöscht.
5. UPDATE-Zugriff auf *Bestellungen*: Aktualisierung von AnzPosten und BestSumme.
6. UPDATE-Zugriff auf *Lieferanten*: Aktualisierung von BestSummeGesamt.
7. COMMIT für alle Änderungsoperationen.

Bei dieser Vorgehensweise sind also insgesamt sieben (entfernte) Datenbankaufrufe erforderlich. [2]

Alternative: Stored Procedure

Naheliegende Funktionsaufteilung in diesem Fall: [1)]

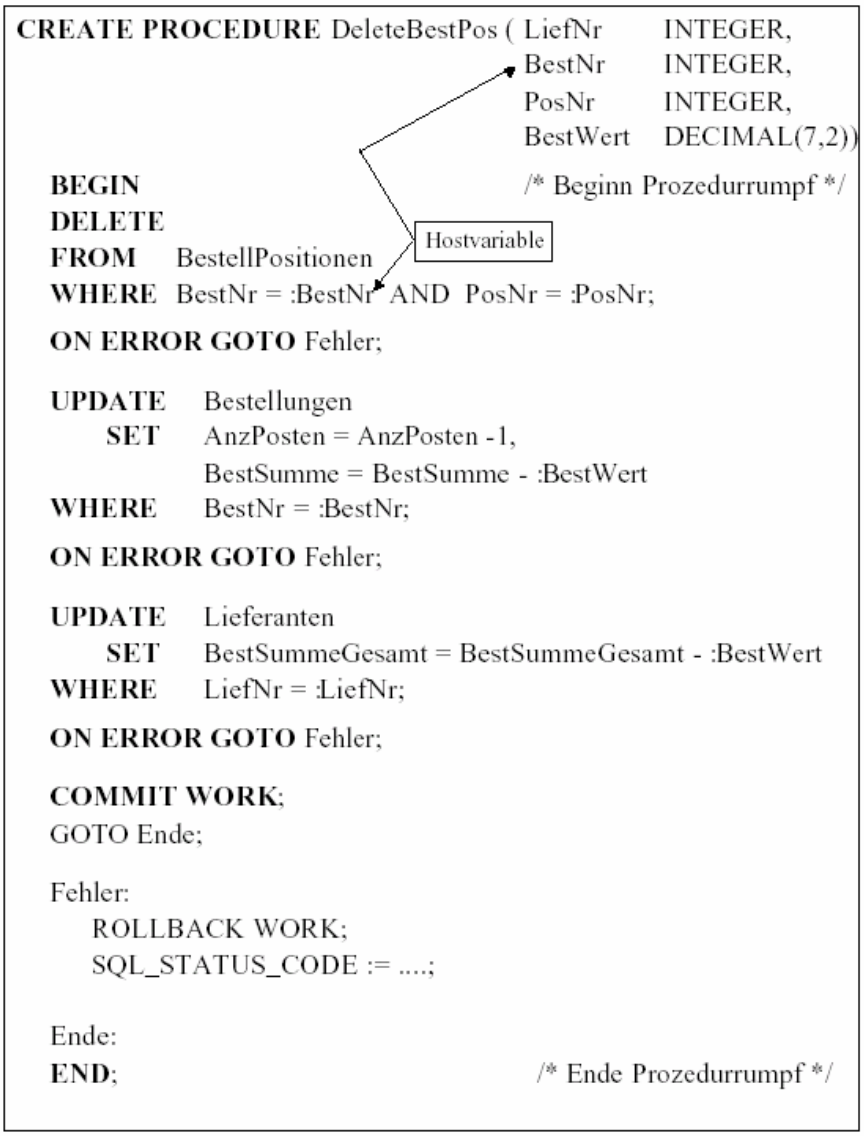
Zusammenfassung der DELETE- und UPDATE-Anweisungen in *einer* Stored Procedure
⇒ DeleteBestPos(LiefNr, BestNr. BestPos, BestWert)

Damit:

Schritte 1 - 3 wie oben

Schritte 4 - 7 als Stored-Procedure-Aufruf

1) Durch Verwendung eines Joins könnten zwar die ersten drei Aufrufe im Prinzip zu einem Aufruf zusammengefaßt werden, damit würde sich allerdings das zu übertragende Datenvolumen stark aufblähen und auch die Verarbeitung im Anwendungsprogramm (Zerlegen des Join-Resultats in die Bestandteile „Lieferant", „Bestellung" und „BestellPositionen") komplizierter werden, so daß man dies in der Regel nicht tun wird.



Alternative dazu: Trigger

Vorgang: Schritt 4 wird über Client ausgeführt, der Rest der Updates übe Trigger

Ablauf hier:

Schritte 1- 4 wie gehabt

Schritt 5 (faßt mittels Trigger die Schritte 5-7 zusammen)

```
CREATE TRIGGER AfterDeleteBestPos
  AFTER DELETE ON BestellPositionen
  FOR EACH ROW
     DECLARE
        LiefNr  Integer;            /* Deklaration Hilfsvariable   */
  BEGIN
  SELECT   LiefNr                   /* Ermittlung LieferantenNr    */
  INTO     :LiefNr                  /* und Speicherung in LiefNr   */
  FROM     Bestellungen
  WHERE    BestNr = old.BestNr;     <-- Zugriff auf alten Tupelwert
  ON ERROR GOTO Ende;
  UPDATE   Bestellungen             /* Aktualisierung Bestellungen */
     SET   AnzPosten = AnzPosten - 1,
           BestSumme = BestSumme - old.BestWert
  WHERE    BestNr = old.BestNr;
  ON ERROR GOTO Ende;
  UPDATE   Lieferanten              /* Aktualisierung Lieferanten  */
     SET   BestSummeGesamt =
           BestSummeGesamt - old.BestWert
  WHERE    LiefNr = :LiefNr;
  Ende:
  END;
```

Zu Schnitt 4:

- Die *gesamte Datenverwaltungsfunktionalität* wird auf dem *Serversystem* bereitgestellt.
- Die *gesamte Applikationsfunktionalität* liegt auf dem *Client-System*.

- Der Client bedient sich hierbei einer semantisch „hohen“ Schnittstelle, wie z. B. SQL oder eines objektorientierten Pendants.
- Die physischen Aspekte des Zugriffs, insbesondere der gezielte Zugriff auf Datenbankseiten, Indexe u. ä. bleiben hierdurch vor dem Client verborgen.
- Im einfachsten Fall bedient sich das Clientsystem der „normalen“ Datenbankschnittstelle des eingesetzten DBMS, wie z. B. Embedded SQL.
- Für die Anwendung ist es in diesem Fall völlig transparent, ob die Datenverwaltungsfunktionalität auf demselben Rechner oder auf einem entfernten Rechner angeboten wird oder ob es sich um den Zugriff auf ein verteiltes DBMS handelt.
- Viele DBMSe bieten über diese Schnittstelle auch den transparenten Zugriff auf Daten anderer Datenbanken desselben oder anderer Hersteller an
  ⇒ ***Database Gateways*** (proprietäre Lösung!)
- Das eigene Server-DBMS fungiert dabei quasi als "Vermittler" (

Typische Ausprägungen und Architekturen von Datenbanksystemen

Multi-Tier-Architekturen von C/S

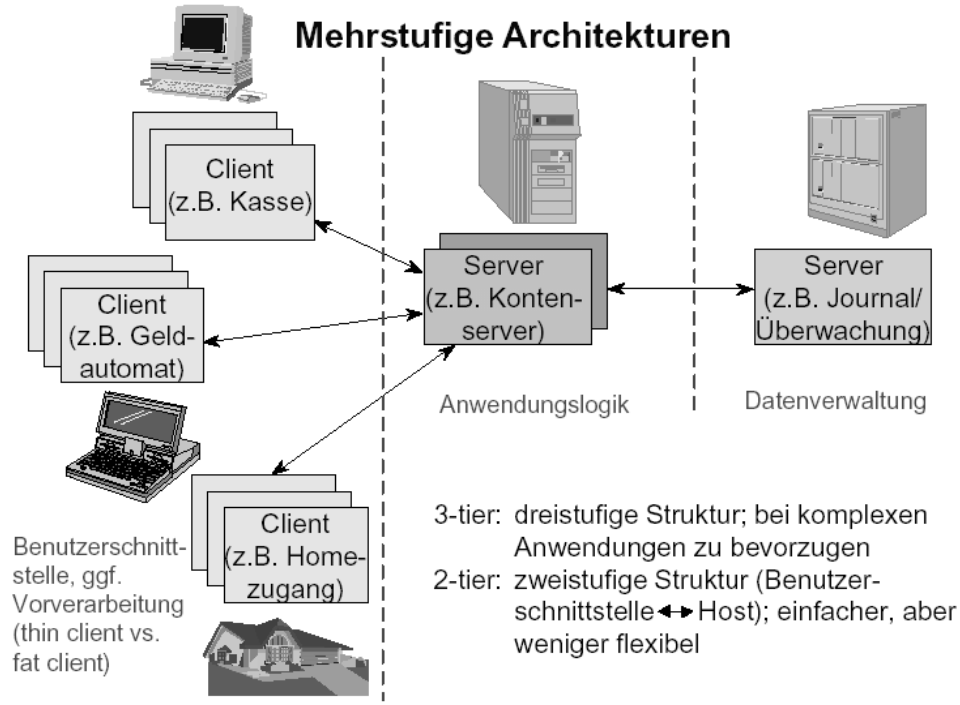
- Schichtenarchitektur:
  - Jede Ebene implementiert einen anderen Aspekt (Datenbank, Anwendungen, GUI, ...)
  - Unterschiedliche Anbieter für einzelnen Schichten (Oracle für die Datenbank, sd&m für die Anwendung, Apache für den Webserver, Microsoft fürs GUI)
- Jede Schicht kann auf einem eigenen Rechner implementiert werden. Es können aber auch mehrere Schichten auf einem Rechner installiert werden.
- Skalierbarkeit auf jeder Schicht bis auf Datenbank.

2- tier- Architektur

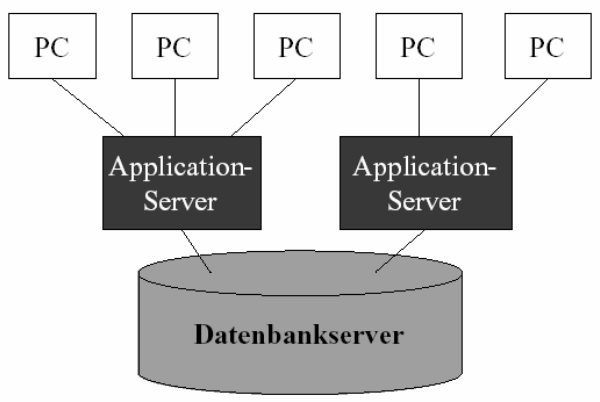
Diese Architektur wird auch als einstufige Client-Server-Topologie bezeichnet. Hierbei gibt es nur eine Schnittstelle, also eine direkte Verbindung, zwischen dem Server und dem Client. Bei der Zwei- tier- Architektur läuft auf dem Client die Präsentation und auf dem Server die Datenbank, die Logikeinheit kann sich entweder auf dem Server oder dem Client befinden. Es gibt verschiedene Formen dieser Architektur, abhängig davon wie viele Server oder Clients verbunden sind, unterscheidet man Single- oder Multi- Server/Client Systeme.

3- tier- Architektur

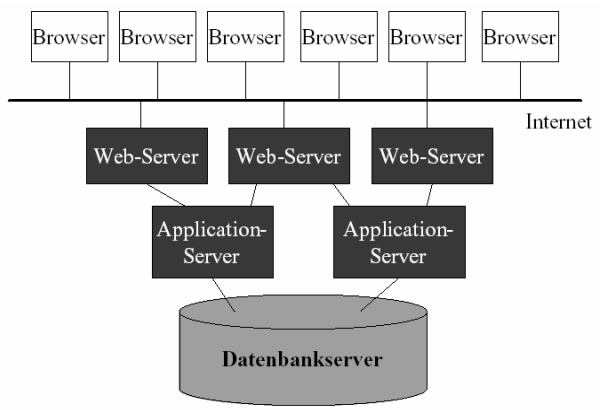
Bei der 3-tier Architektur handelt es sich ebenfalls um eine C/S Architektur bei der allerdings drei Servertypen auf jeweils einer Ebene zum Einsatz kommen. Diese Ausprägung mit Datenbank-, Applikations- und Präsentationsserver bzw. Client ist die gängigste C/S Architektur.



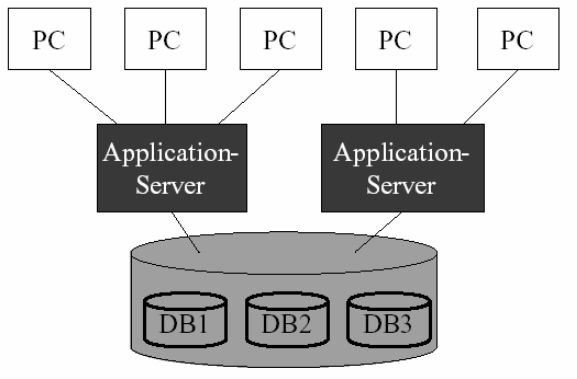
Three-Tier



Datenbanken im WEB

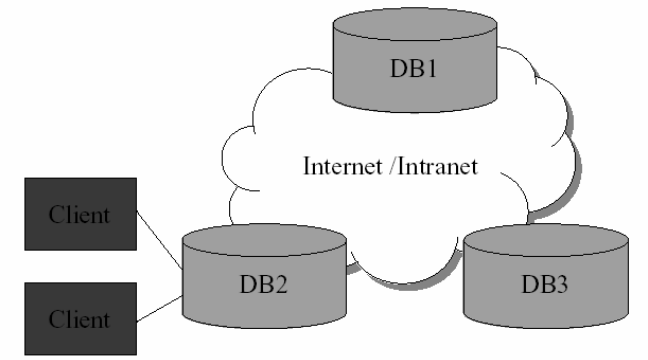


Parallele Datenbanksysteme



- Das Datenbanksystem selber ist aus mehreren Knoten (Festplatten, CPUs) aufgebaut. Die Knoten sind durch ein schnelles Netzwerk verbunden.
- Ziele:
  - Höheren Durchsatz (Inter-Query Parallelität)
  - Niedrigere Antwortzeiten (Intra-Query Parall.)
  - Höhere Verfügbarkeit
  - Kosten, Erweiterbarkeit, Skalierbarkeit
- Wichtig: Transparenz

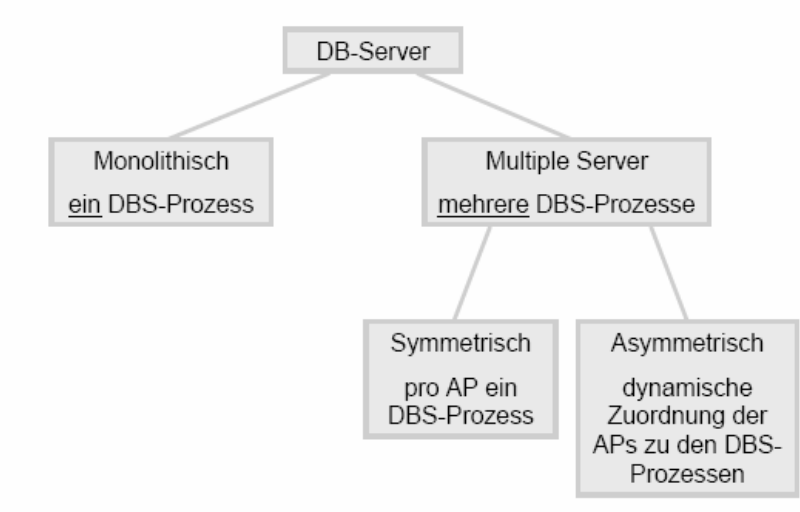
Verteilte Datenbanksysteme



- Die einzelnen Datenbanken sind autonom und durch ein langsames, instabiles Netzwerk verbunden. Zugriff von überall auf alles möglich.
- Großen Organisationen mit vielen Filialen.
- Prinzip: Speichere Daten, wo sie gebraucht werden. Eventuell: Replikation.
- Transparenz:
  - Benutzer weiß nicht, wo Kopien, welcher Daten liegen. Aus Sicht des Benutzers ist der Zugriff auf das gesamte System wie bei einem zentralen System.

11.2 Oracle Netzwerk

Von der Konstruktion her unterscheidet man folgende Serverkonstruktionen:



Multiple Server

(1) mehrere DB Serverprozesse

Kommunikation zwischen den Servern über shared memory
zwischen Clients und Servern bzw. Dispatcher über BS-Mechanismen (IPC) oder Netzsoftware

Symmetrische Zuordnung - jedem Client ist genau ein Server Prozess zugeordnet: statische Zuordnung, feste Anzahl n Server vorgeneriert

==> maximaler Parallelitätsgrad ist n

Asymmetrisch - ein Client wird mit Hilfe eines Dispatcher mit einem Server Prozess verbunden. Feste Anzahl Server vorgeneriert, aber Parallelitätsgrad kann höher sein

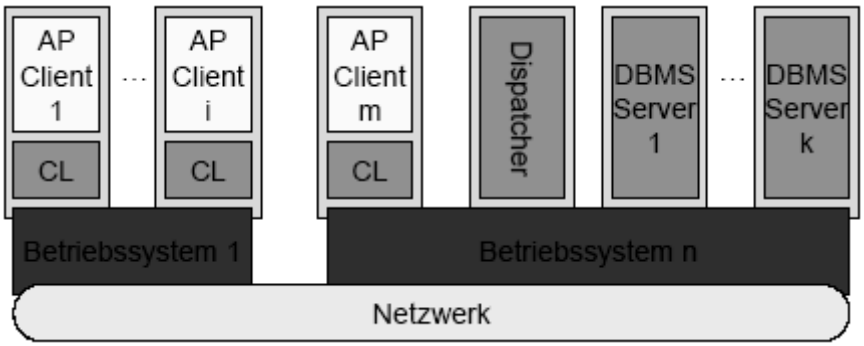
(2) ein DB Serverprozess

Synchronisierter Zugriff auf Systempuffer und zentrale Systemtabellen
Server verwendet Multi-Threading (Reentrant Code)
genau ein Server-Prozess für viele Clients
DB-Serverprozess ist vom BS bevorzugt!

Allgemeine Eigenschaften:

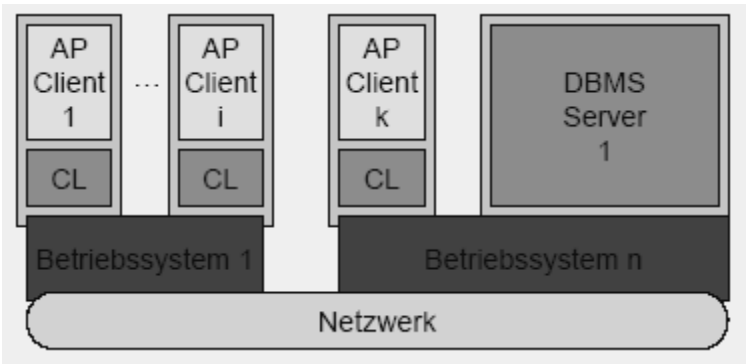
DBMS ist ein Verbund von verschiedenen Prozessen
Kommunikation über Betriebssystem (z.B. IPC) oder Netzwerk
Scheduling der Prozesse geschieht über das Betriebssystem, grosser Vorteil bei Mehrprozessor-Rechnern, da Prozessorzuteilung durch das BS erfolgt

z.B. Oracle, Informix, DB2



Monolithische Server

eigene Ressourcenverwaltung, dupliziert BS-Funktionen
einfache Kommunikation im Server durch Shared Memory
z.B. Sybase, MS SQL Server



Beispiele:

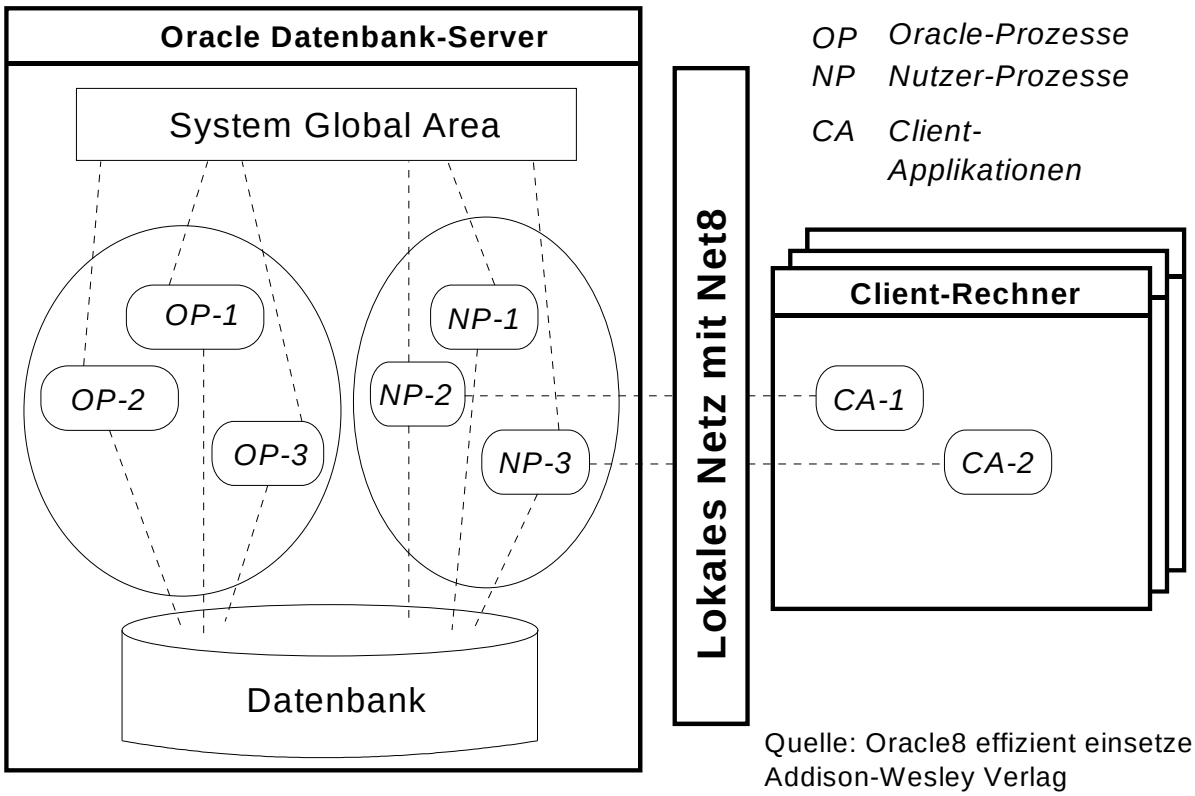
Sybase: Monolithischer Server

ein Serverprozess
eigenes Process -Scheduling
weniger Shared Memory

Oracle: Multipler Server

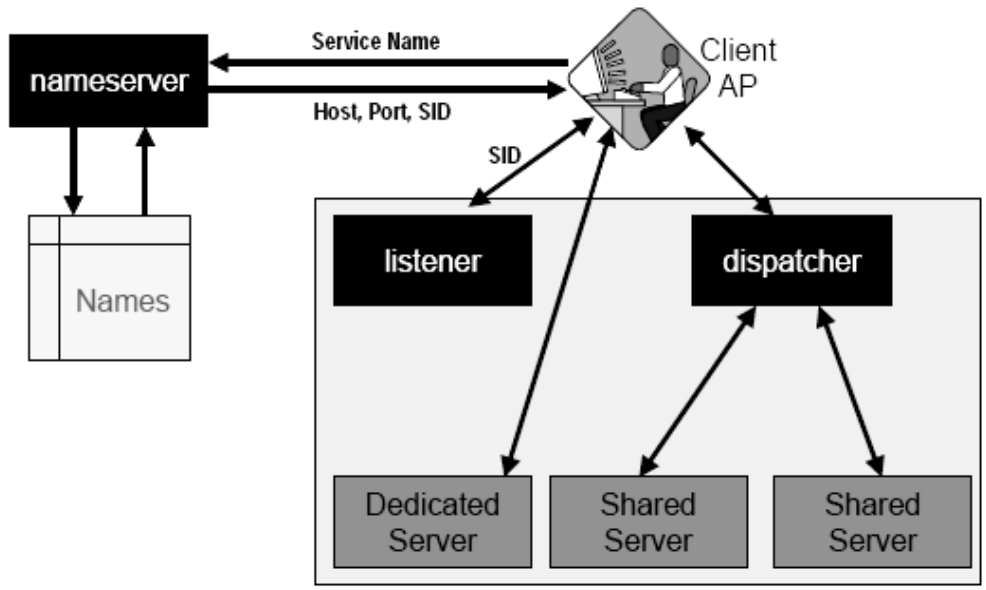
sowohl symmetrisch als auch asymmetrisch konfigurierbar
dedicated Server
shared Server mit Dispatcher

DB-Prozesse mit verschiedenen Aufgaben:
Recoverer, Process Monitor,System Monitor, Database Writer, Log Writer, Archiver, Checkpoint, Dispatcher, Lock



Quelle: Oracle8 effizient einsetzen
Addison-Wesley Verlag

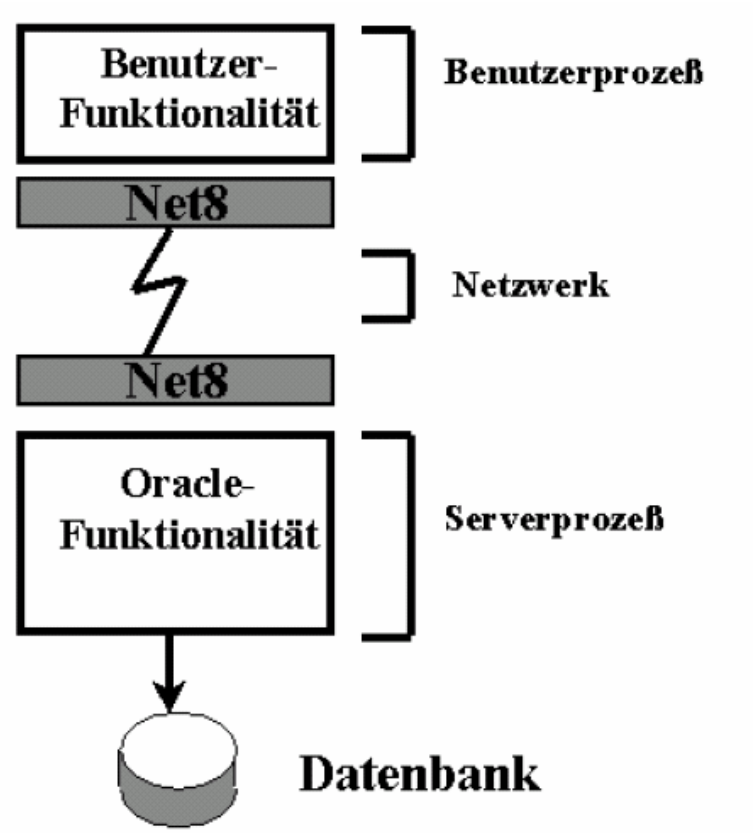
Oracle Verbindungsaufbau:



Net8 ist die Netzwerk-Software, die für die Kommunikation zwischen Cleint und Server in einer Oracle-Umgebung benötigt wird.

Eine Net8-Verbindung besteht im einfachen Fall aus einer Netzwerkverbindung eines beliebigen Clients (z.B. einem SQL*Plus-Programm, das auf einem PC läuft) mit einem Oracle8 Serverprozess über ein Netzwerkprotokoll (z.B. TCP/IP).

Sobald die Verbindung besteht, tauschen Client und Server ausschließlich auf direktem Weg Informationen aus.



Eine besondere Rolle kommt dem **Transparent Network Substate (TNS)** zu.

Um die begrenzten Netzwerk-Ressourcen optimal zu nutzen werden inaktive Verbindungen temporär anderen Clients zur Verfügung gestellt (Connection Pooling)

Durch Multiplexing können darüber hinaus auch mehrere Verbindungen über eine physische Verbindung gebündelt werden.

TNS-Listener

Der TNS-Listener ist ein Prozess, der die Verbindungsanfragen erkennt, die Adressdaten übermittelt und die Verbindung herstellt.

Beim Verbindungsaufbau kommt (mindestens) ein weiterer Prozess ins Spiel - der **Net8 Listener**. Dies hat folgenden Hintergrund: Client und Server kommunizieren über Netzwerkadressen, die zwar für die Dauer der Verbindung fest sind, beim Verbindungsaufbau aber dynamisch vergeben werden. Im Falle von TCP/IP haben zwar beide eine eindeutige IP-Adresse, die eingehende Kommunikation läuft, aber sowohl beim Client als auch beim Server über einen Port, der für die Lebenszeit der Verbindung dynamisch festgelegt wird. Beim Verbindungsaufbau wird also ein Ansprechpartner mit einer festen

Netzwerkadresse benötigt, der vom Client aus aktiviert werden kann. Diese Rolle übernimmt der Listener.

TNS Adressierung

Die Adressierung im Netz erfolgt anhand von IP-Adressen z.B. 102.54.54.97 und einem Namen z.B. rhino.acme.com . Alle in einem Netz adressierbaren Server und Clients sind im hosts -File gespeichert.

TNS-Konfiguration

Die Konfiguration des Netzwerks besteht nun aus dem Erzeugen und Verteilen entsprechender Parameterdateien für Client und Server. Im einzelnen sind dies:

Datei	Client/Server	Inhalt
Sqlnet.ora	C/S	allgemeine Parameter
Listener.ora	-/S	Netzwerkabhängige Adressinformation sowie verfügbare Instanzen
Tnsnames .ora	C/S	Net8-Namen und zugehörige Adressinformationen

Die Datei tnsnames.ora hat zwar Client-Charakter, wird aber dennoch auf dem Server benötigt, wenn vom Server aus z.B. mit SQL*Plus auf andere Instanzen zugegriffen wird oder wenn Datenbanklinks benutzt werden. Im allgemeinen müssen die Dateien sqlnet.ora und tnsnames.ora einmalig erzeugt und für alle Installationen von Net8 verteilt werden; eine Datei listener.ora wird für jeden Server benötigt.

Die Dateien werden von der Oracle-Software standardmäßig in bestimmten Verzeichnissen erwartet ($ORACLE_HOME$\NET80\ADMIN).

Die Erstellung der Datei(en) listener.ora steht meist am Anfang der Konfiguration. (Vergleiche dazu NET8-Assistent)

Die Listener-Konfiguration kann mit dem Net8 Assistant erzeugt werden. Für die Konfiguration sind zwei Elemente notwendig: die Listener-Adressen und die Instanzen, für die Requests entgegengenommen werden. Zunächst muss im Net8 Assistant auf der linken Seite der Eintrag Listener ausgewählt werden. Durch das Anklicken des +-Symbols wird ein neuer Listener-Eintrag erzeugt. Als Name für den Listener empfiehlt sich LISTENER:
Bei Administrationskommandos muss der Name des Listener angegeben werden, es sei denn, der Name ist LISTENER. Eine Listener-Adresse kann dann im Bereich Listening-Adressen konfiguriert werden.

Es empfiehlt sich, für TCP/IP den Port 1521 (seltener: 1526) zu konfigurieren, da diese beiden Ports für Oracle-Kommunikation reserviert sind. Nun kann die

Instanz im Bereich Datenbankdienste konfiguriert werden. Einzige Pflichtangabe ist hier die SID; in diesem Beispiel lautet die SID = MYDB.

Alle anderen Einstellungsmöglichkeiten belassen Sie bei den Defaultangaben. Sobald die Netzwerkkonfiguration gespeichert wird erhält man die Datei listener.ora:

Beispiel:

```
# C:\ORANT\NET80\ADMIN\LISTENER.ORA
# Configuration File:C:\orant\net80\admin\listener.ora
# Generated by Oracle Net8 Assistant
PASSWORDS_LISTENER= (oracle)
LISTENER =
(ADDRESS_LIST =
.............
(ADDRESS = (PROTOCOL = TCP)(HOST = 141.71.22.59)(PORT = 1521))
(ADDRESS = (PROTOCOL = TCP)(HOST = 141.71.22.59)(PORT = 1526))
.............
)
SID_LIST_LISTENER =
(SID_LIST =
(SID_DESC =
(SID_NAME = extproc)
(PROGRAM = extproc)
)
(SID_DESC =
(SID_NAME = MYDB)
)
)
```

Konfigurationsvarianten von Oracle8
:

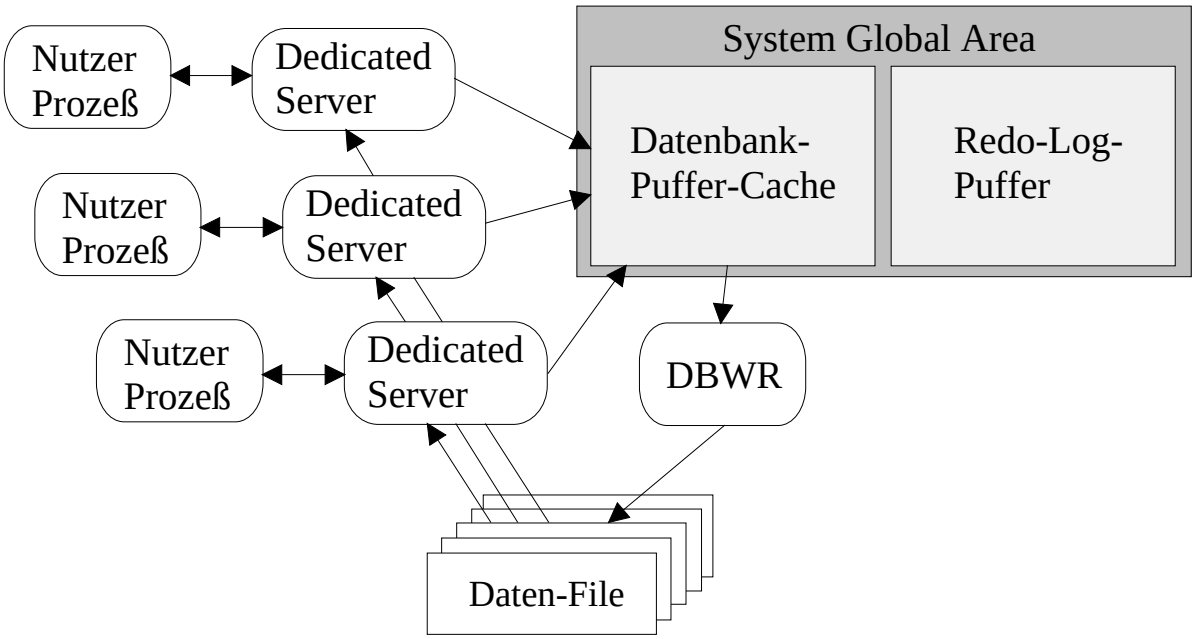
1. Single-Prozeß-Konfiguration:

- in der Praxis heutzutage kaum noch anzutreffen
- unteres Ende des Skalierungsspektrums von Oracle8
- nur 1 Nutzer kann auf das O8-System und die DB zugreifen
- keine nebenläufigen Prozesse
- Applikationscode und DBMS-Code bilden einen Block

2. Dedicated-Server-Konfiguration:

- Standardkonfig. für kleine u. mittlere O8-Anwendungen
- geeignet für Anwendungen mit großer Datenbanklast
- Rechner mit O8-Instanz benötigt Multitask.-Betriebssystem
- jedem Client steht ein dedizierter Server-Proz. zur Verfügung
- schnelle Auslastung der Systemressourcen

Skizze einer Dedicated-Server-Konfiguration:

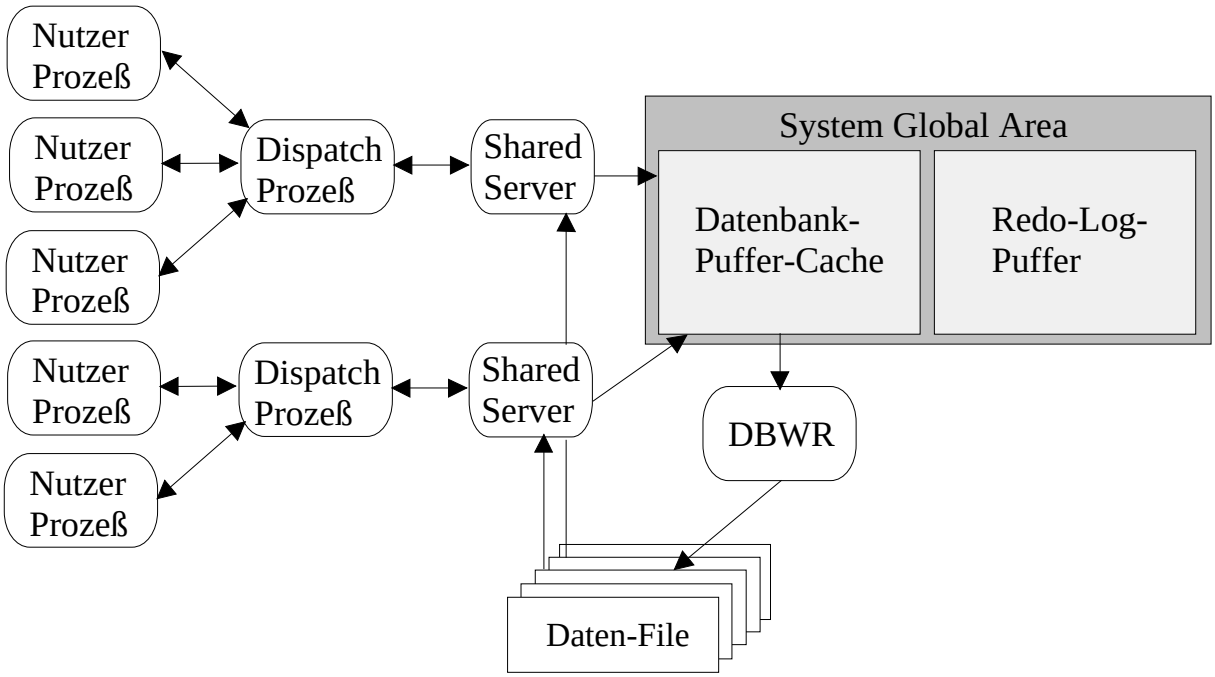


Konfigurationsvarianten von Oracle8:

3. Multi-Threaded-Server-Konfiguration:

- geeignet für Anwendungen mit geringerer Datenbanklast
- ein Multi-Threaded-Server bedient mehrere Client-Prozesse
- Zugriff wird über Dispatcher-Prozeß gesteuert
- Verbindung zwischen Clients u. Server erfolgt in Zeitscheiben
- bei zu großer DB-Last wird Client-Prozeß schlecht bedient

Skizze einer Multi-Threaded-Server-Konfiguration:



4. Parallel-Server-Konfiguration:

- unterstützt lose und eng gekoppelte Rechnersysteme
- mehrere Rechner o. Prozessoren arbeiten mit einer log. DB
- Leistungssteigerung durch parallelisierte Anfragebearbeitung
- Synchronisationen werden durch Hintergrundprozesse gelöst
- Erhöhung der Verfügbarkeit der Datenbank