Oracle Performance Tuning

Alex Schröder

2001-06-29

Inhaltsverzeichnis

Kapitel 1

Einleitung

Performance ist ein subjektives Mass für die Geschwindigkeit, mit der eine Applikation verwendet werden kann. Die Bewertung der Performance kann nur relativ sein: Das System ist schneller als vorher, als auf einer anderen Maschine, als mit der alten Version.

Tuning ist eine Tätigkeit, welche die Performance steigert. Diese Tätigkeit wird vor, während und nach der Entwicklung einer Applikation ausgeführt. Vor dem Start der Entwicklung muss das Design der Applikation die Performance ermöglichen, die der Kunden erwartet. Während der Entwicklung muss die Performance unter möglichst realistischen Bedingungen überprüft werden. Nach der Auslieferung der Software muss die Performance punktuell verbessert werden können.

Realistische Bedingungen lassen sich nur schwer herbeiführen, wenn man nicht beim Kunden arbeitet. In diesem Fall beschränkt man sich oft auf eine realistische Menge an *Testdaten*. Existiert schon eine Datenbank, kann man Belastung von Server und Netzwerk schon mit berücksichtigen. Dies ist meist nur dann gegeben, wenn das System schon im produktiven Einsatz ist.

1.1 Testatkriterien

Jede Stunde werden zwei Studenten *einen kurzen Vortrag* halten. Nach jeder Doppelstunde werde ich mit den betreffenden Studenten ein Thema vereinbaren. Es wird sich im Normalfall um die Erklärung von Übungen handeln, welche in der Stunde nicht mehr behandelt werden können, oder es wird sich um Begriffe handeln, welche in der Stunde aufgetaucht sind.

Der Inhalt des Vortrages soll *zusätzlich in elektronischer Form* abgegeben werden. Im Umfang sollte dieses Dokument zwei oder drei Paragraphen Text sowie eine oder zwei erklärende Grafiken nicht überschreiten. Idealerweise werden die Dokumente auf dem Web veröffentlicht.

Die benötigte Information darf natürlich aus allen möglichen Quellen stammen (Bücher, Web Seiten). Diese Quellen sind zu referenzieren.

1.2 Prüfung

Dieses Skript enthält mögliche Prüfungsfragen. Diese und ähnliche Fragen werden an der Prüfung gestellt.

An den Prüfungen dürfen beliebige schriftliche Unterlagen verwendet werden.

1.3 Zum Skript

Dieses Skript ist ein dynamisches Dokument. Falls sie Fehler darin entdecken, oder falls einzelne Fragen ihnen vom Thema oder vom Schwierigkeitsgrad nicht angemessen erscheinen, bin ich für jeden Feedback dankbar. Schicken sie mir eine Mail an asc@bsiag.com.

Die Informationen in diesem Skript basieren auf den folgenden Büchern:

- *Oracle7 Server Concepts* (1995), eines der offiziellen Oracle Handbücher, wurde für die Beschreibung der Speicherstrukturen und der Prozess Architektur verwendet.
- *Fundamentals of Database Systems* von Ramez Elmasri und Shamkant B. Navathe (2000) wurde für die Beschreibung der B*Tree Indexe und für die Speicherhierarchien (Primär, Sekundär, Tertiär) verwendet.
- *ORACLE Performance Tuning* von Peter Corrigan und Mark Gurry (1993) wurde für SQL Tuning Tipps und Oracle Speicher Tuning verwendet.
- Die Kursunterlagen *K3510 V1.1(C) - Oracle7 Performance Tuning* (1998) wurden für die Optimierung der SGA Parameter verwendet.
- *Oracle 7.3.3 Server Reference Manual* (1996) und *Oracle 8.1.5 Reference* (1999), bieten eine Beschreibung aller Parameter für die Control Files (INITxxxx.ORA files), eine Beschreibung aller Static Data Dictionary Views (z. B. ALL_VIEWS) und aller Dynamic Performance Views (z. B. V$PARAMETER), und vieles mehr.
- *Oracle 7.3.3 Tuning* und *Oracle 8.1.5 Tuning* (1999) bieten alle Tuning Informationen, die das Herz begehrt, in ausführlicher Form.

1.4 Ziel

Um die Performance zu steigern, werden wir ein gutes Design verwenden, effiziente SQL Statements schreiben, und die Datenbank richtig aufsetzen. *So erreichen wir, dass auf der Datenbank weniger Speicher benötigt wird und die Festplattenzugriffe reduziert werden können.*

Design Der Aufbau der Datenbank bestimmt die möglichen SQL Statements, welche verwendet werden können.

Applikation Der Aufbau der Applikation bestimmt die SQL Statements, die zu schreiben sind.

Speicher Die Menge an physischem Speicher bestimmt, wieviel Arbeit Oracle ohne Festplattenzugriffe leisten kann.

I/O Die Anzahl Zugriffe auf die Festplatten und die Verteilung der Festplatten bestimmen, wie stark Oracle durch Festplattenzugriffe gebremst wird. Der Netzwerkverkehr und die Bandbreite des Netzes beschränken die Kommunikation zwischen Server und Client.

Contention Eine grosse Anzahl Benutzer und eine grosse Anzahl Prozesse können zu Locking Problemen führen.

Mögliche Prüfungsfragen

- Nennen sie die fünf grossen Aspekte von Performance Tuning. Beurteilen sie die einzelnen Aspekte bezüglich den entstehenden Kosten.
- Nennen sie die fünf grossen Aspekte von Performance Tuning. Zeigen sie auf, wann die einzelnen Aspekte im Rahmen eines Projektplanes berücksichtigt werden müssen.
- Nennen sie die fünf grossen Aspekte von Performance Tuning. Welche Abklärungen sind vom Projektleiter zu treffen, um ein möglichst gutes Bild von den zu erwartenden Problemen zu haben?

Kapitel 2

Architektur

Um zu verstehen, warum gewisse Massnahmen dazu beitragen, die Performance einer Applikation zu steigern, werfen wir einen kurzen Blick auf die Oracle Architektur. Hier soll kurz erklärt werden, wie die Daten auf der Festplatte gespeichert werden und wie Oracle auf diese Daten zugreift. Inwiefern all diese Komponenten dann die Performance beeinflussen können, werden wir in den anschliessenden Kapiteln sehen.

Zuerst eine Einführung am System. Die benötigten Skripts befinden sich im sql Unterverzeichnis.

- Wir werden alle gleichzeitig Daten in die Oracle Datenbank speichern. Werden die Daten *gleichzeitig oder nacheinander* geschrieben?
- Wir schauen uns eine Alternative an: Den *SQL*Loader*. Performance bei kleinem Experiment daheim: insert-script mit SQL*Plus für Anfangsbuchstabe A dauert 3:35 Minuten, load-script mit SQL*Loader für Anfangsbuchstabe A dauert 3.4 Sekunden. Der SQL*Loader kann sogar Datablocks direkt schreiben. Performance bei kleinem Experiment daheim: load-script für Anfangsbuchstabe A wie vorher dauert 3.9 Sekunden, load-script für Anfangsbuchstabe A mit *direct path* dauert 1:6 Sekunden.
- In der View USER_TABLES finden wir mehr Informationen zu unserer Tabelle: Wieviele Datenblöcke werden verwendet, wieviele sind noch frei, wieviele Extents werden verwendet, wie gross wird das nächste Extent. Falls die Infos fehlen, müssen diese mit einem ANALYZE TABLE zuerst berechnet werden.
- Ändert man die Daten, werden die Statistiken nicht nachgeführt. Beachten sie, dass ein Update, welches den benötigten Platz pro Datensatz erhöht, zu Block Chaining führen kann. Hierzu später mehr.

2.1 Zugriff auf den Datadictionary

Die gesamte Information, die Oracle über die Daten besitzt, liegt im Data Dictionary. *Als Benutzer hat man Zugriffe auf Teile dieser Information*, als DBA hat man Zugriff zu all diesen Informationen. Die Daten liegen in V$ Views.

Die V$* Views sind Synonyme für V_$* Views. Die Namen stehen in der V$FIXED_TABLE. Auf diese Fixed Tables gibt es wiederum zusammenfassende Views. Diese kann man sich aus der View ALL_VIEWS auslesen. Views mit Informationen für alle Benutzer beginnen mit ALL, Views mit Informationen für den momentan angemeldeten Benutzer beginnen mit USER und Views mit Informationen für den Database Administrator beginnen mit DBA. Weitere Informationen hierzu finden man im *Oracle 8.1.5 Reference* Handbuch.

Beispiel: Wie finden sie alle Data Dictionary Views, welche mit den eigenen Indexen und den Spalten in diesen Indexen zu tun haben?

```
SQL> select view_name
     from all_views
     where view_name like 'USER%INDEX%'
     or view_name like 'USER%COLUMN%';

VIEW_NAME
------------------------------
USER_CLU_COLUMNS
USER_CONS_COLUMNS
USER_INDEXES
USER_INDEXTYPES
USER_INDEXTYPE_OPERATORS
USER_IND_COLUMNS
USER_PART_INDEXES
USER_PART_KEY_COLUMNS
USER_SUBPART_KEY_COLUMNS
USER_TAB_COLUMNS
USER_UPDATABLE_COLUMNS

11 rows selected.
```

Die Views USER_INDEXES und USER_IND_COLUMNS scheinen hier die relevanten Views zu sein.

2.1.1 Tabellen und Spalten

Um gewisse Spalten in DBA_TABLES und DBA_TAB_COLUMNS zu füllen, wird der ANALYZE TABLE Befehl verwendet. Hiermit erhält man Informationen über die Anzahl Zeilen (rows), sowie über den benötigten Platz (blocks).

Natürlich ist klar, dass für die eigenen Tabellen die Views USER_TABLES und USER_TAB_COLUMS, bzw. für die öffentlich einsehbaren Tabellen die Views ALL_TABLES und ALL_TAB_COLUMNS verwendet werden.

```
SQL> select table_name, num_rows, blocks,
            empty_blocks, last_analyzed
     from user_tables
     where table_name = 'PERSON';

TABLE_NAME           NUM_ROWS     BLOCKS EMPTY_BLOCKS LAST_ANAL
-------------------- -------- ---------- ------------ ---------
PERSON
```

```
SQL> analyze table person compute statistics;

Table analyzed.

SQL> select table_name, num_rows, blocks,
            empty_blocks, last_analyzed
     from user_tables
     where table_name = 'PERSON';

TABLE_NAME          NUM_ROWS     BLOCKS EMPTY_BLOCKS LAST_ANAL
----------------- ---------- ---------- ------------ ---------
PERSON                     2          1            3 19-NOV-00
```

Die neu gewonnene Information in der Tabelle ALL/USER/DBA_TAB_-COLUMNS beschreibt die Datenverteilung in den einzelnen Spalten und kann vom Optimizer verwendet werden. Der Optimizer generiert den Execution Plan für ein SQL Statement. *Es gibt zwei Optimizer bei Oracle: Rule-Based und Cost-Based.* Der Cost-Based Optimizer (CBO) kann die Informationen über die Datenverteilung verwenden, um zwischen zwei verschiedenen Execution Plans entscheiden zu können.

```
SQL> select substr(column_name,1,20), num_distinct,
            num_nulls, last_analyzed
     from user_tab_columns
     where table_name = 'PROJEKT';

SUBSTR(COLUMN_NAME,1 NUM_DISTINCT  NUM_NULLS LAST_ANAL
-------------------- ------------ ---------- ---------
PROJEKTNR
NAME
KUNDENR
STATUS

SQL> analyze table projekt compute statistics;

Table analyzed.

SQL> select substr(column_name,1,20), num_distinct,
            num_nulls, last_analyzed
     from user_tab_columns
     where table_name = 'PROJEKT';

SUBSTR(COLUMN_NAME,1 NUM_DISTINCT  NUM_NULLS LAST_ANAL
-------------------- ------------ ---------- ---------
PROJEKTNR                       3          0 19-NOV-00
NAME                            3          0 19-NOV-00
KUNDENR                         3          0 19-NOV-00
STATUS                          2          0 19-NOV-00
```

2.1.2 Platzverbrauch

In der Tabelle V$PARAMETER stehen die Datenbank Parameter, welche für das laufende System gelten. Unter anderem findet man dort auch den Parameter 'db_block_size', denn man verwenden kann:

```
SQL> select p.value * u.blocks / 1024 KB
     from v$parameter p, all_tables u
     where p.name = 'db_block_size'
     and u.table_name = 'TEXTE';

        KB
----------
      4818
```

2.2 Die Oracle Speicherstrukturen

Die Daten werden auf die Festplatte in *Blocks* geschrieben und von der Festplatte in Blocks gelesen. Die Grösse eines Blocks oder Datablocks ist ein Vielfaches der Block Grösse auf der Festplatte. Ein Block ist also meistens 2KB oder 4KB gross, kann aber 32KB oder grösser sein. Der Datenbank Cache basiert auch auf Datablocks.

Je grösser die Blocks, desto schneller können Indexe gelesen werden. Das liegt daran, dass B-Tree Leaves Blocks sind, je grösser die Blocks also sind, umso weniger Leaves gibt es im Index, dh. der Index wird entsprechend kleiner.

Eine Reihe zusammenhängender Datablocks bildet ein *Extent*. Wenn der für eine Tabelle reservierte Platz verbraucht wurde, versucht die Datenbank, einen neuen Extent anzulegen. Die Grösse des Extents kann bei der Erstellung der Tabelle angegeben werden.

Der für eine Datenstruktur (Tabelle, Index) reservierte Platz nennt man ein *Segment*. Ein Segment besteht aus einem oder mehreren Extents. Segmente können auch für Datenstrukturen verwendet werden, welche nicht direkt zu Objekten gehören, beispielsweise Rollback Segmente.

Es gibt vier Sorten von Segmente:

1. Tabellen Segmente
2. Index Segmente
3. Rollback Segmente
4. Temporäre Segmente

Die Segmente wiederum werden in *Tablespaces* zusammengefasst. Falls bei der Erstellung von Tabellen und Indexen die Grösse des ersten Extents nicht angegeben wurde, werden die default Werte des Tablespaces verwendet. Diese default Werte kann man bei der Erstellung von Tablespaces angeben.

Die Datenbank benötigt mindestens einen Tablespace, den *System Tablespace*. Im System Tablespace ist der *Data Dictionary* abgelegt. Die restlichen Daten sollten in weiteren Tablespaces abgelegt werden.

Der Tablespace ist die logische Einheit, welche der Datenbank Administrator (DBA) manipuliert. Ein Tablespace kann *online* (normal zugänglich), *offline*

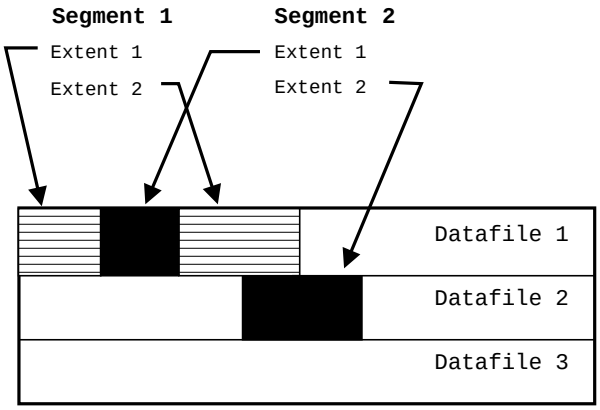

Abbildung 2.1: Struktur eines Tablespaces

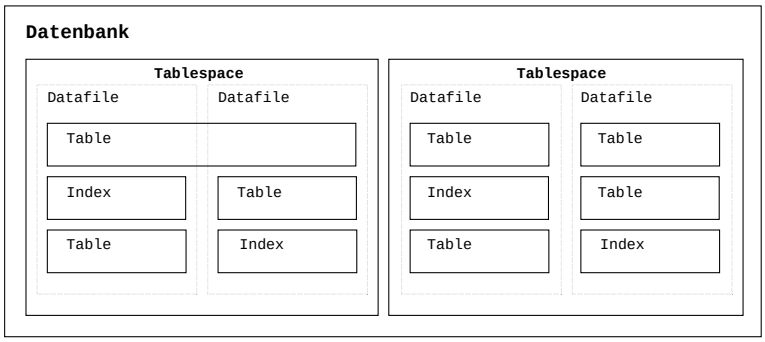

Abbildung 2.2: Struktur der Datenbank

(die Daten sind nicht zugänglich), oder *read-only* (z. B. auf einer CD) sein. *Der System Tablespace muss immer online sein. Der Status aller anderen Tablespaces lässt sich bei laufendem Betrieb ändern.* Benutzer, welche Daten auf offline Tablespaces lesen oder auf read-only Tablespaces schreiben wollen erhalten Fehlermeldungen von der Datenbank.

Ein Tablespace kann aus mehreren *Datafiles* bestehen. Die Datafiles sind schlussendlich die physischen Objekte, in denen die Daten gespeichert werden. Die einzelnen Extents eines Segmentes können auf verschiedene Datafiles im selben Tablespace verteilt sein. Die Datafiles werden, wenn sie angelegt werden, für die Datenbank reserviert. *Das Betriebsystem kann diesen Platz nicht weiter verwenden.* Dies bedeutet, dass ein Datafile durchaus 50MB gross sein kann, selbst wenn nur wenige Bytes an Daten darin gespeichert wurden.

Die Aufteilung der Datenbank in mehrere Tablespaces, und die Aufteilund der Tabellen innerhalb eines Tablespaces in ein Segment und mehrere Extents ist *eine logische Aufteilung* der gespeicherten Daten. Tablespaces besteht aus verschiedenen Datafiles; diese enthalten die schlussendlich die gespeicherten Daten.

Mögliche Prüfungsfragen

- Warum verwendet man SQL*Loader wenn es darum geht, grosse Daten-

mengen aus Text-Dateien in die Datenbank zu importieren? Das Import Programm von Oracle kann auch grosse Datenmengen sehr schnell in eine Datenbank einfügen. Was sind die Nachteile?

- Was ist der Nachteil, wenn man beim SQL*Loader Daten per *direct path* lädt?
- Zeichnen sie schematisch, wie Segmente, Extents und Datenblöcke zusammenhängen. Beschreiben sie, was in diesem Zusammenhang passiert, wenn eine Tabelle neu erstellt wird und dann nach und nach mit Daten gefüllt wird.
- Was sind Oracle Blocks und wie werden sie von der Datenbank verwendet?
- Erklären sie den Zusammenhang zwischen Datensätzen in einer Tabelle und Oracle Blocks.

2.3 Speicherstrukturen einer Oracle DB

Um herauszufinden, welche Tablespaces es gibt, aus welchen Datafiles diese bestehen, und wo sich diese im Filesystem befinden, gehen wir wie folgt vor: Im SQL*Plus steigen wir als DBA ein und untersuchen die Tabellen V$TABLESPACE und V$DATAFILE, dann schauen wir auf dem Filesystem nach (in diesem Fall Linux).

```
SQL> select * from v$tablespace;

       TS# NAME
---------- ------------------------------
         0 SYSTEM
         1 OEM_REPOSITORY
         2 RBS
         3 TEMP
         4 USERS
         5 INDX

6 rows selected.

SQL> select ts#, substr(name,1,40) name,
            status, enabled, bytes
     from v$datafile;

TS# NAME                                     STATUS  ENABLED         BYTES
--- ---------------------------------------- ------- ---------- ----------
  0 /opt/oracle/oradata/perfdb/system01.dbf  SYSTEM  READ WRITE  183500800
  1 /opt/oracle/oradata/perfdb/oemrep01.dbf  ONLINE  READ WRITE    5242880
  2 /opt/oracle/oradata/perfdb/rbs01.dbf     ONLINE  READ WRITE   34797568
  3 /opt/oracle/oradata/perfdb/temp01.dbf    ONLINE  READ WRITE   10485760
  4 /opt/oracle/oradata/perfdb/users01.dbf   ONLINE  READ WRITE   10485760
  5 /opt/oracle/oradata/perfdb/indx01.dbf    ONLINE  READ WRITE   10485760

6 rows selected.


bash-2.04$ ls -l
insgesamt 258676
-rw-r-----   1 oracle   oinstall  4253696 Nov 19 18:05 control01.ctl
-rw-r-----   1 oracle   oinstall  4253696 Nov 19 18:05 control02.ctl
```

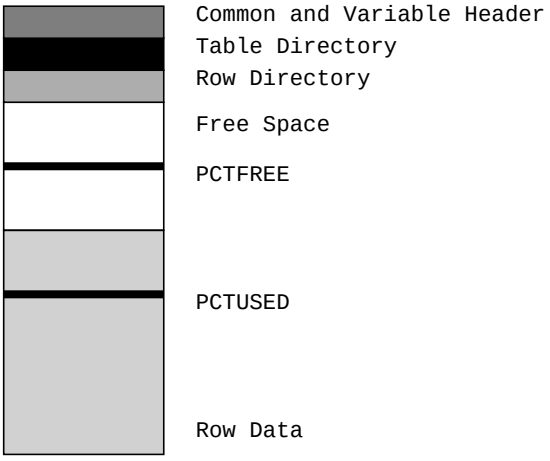

Abbildung 2.3: Aufbau eines Datenblocks

```
-rw-r-----   1 oracle   oinstall 10487808 Nov 19 17:59 indx01.dbf
-rw-r-----   1 oracle   oinstall  5244928 Nov 19 17:59 oemrep01.dbf
-rw-r-----   1 oracle   oinstall 34799616 Nov 19 17:59 rbs01.dbf
-rw-r-----   1 oracle   oinstall   512512 Nov 19 17:59 redo01.log
-rw-r-----   1 oracle   oinstall   512512 Nov 19 18:00 redo02.log
-rw-r-----   1 oracle   oinstall 183502848 Nov 19 17:59 system01.dbf
-rw-r-----   1 oracle   oinstall 10487808 Nov 19 17:59 temp01.dbf
-rw-r-----   1 oracle   oinstall 10487808 Nov 19 17:59 users01.dbf
```

2.3.1 Chained and Migrated Rows

Chained Rows sind Datensätze, welche nicht nicht in einen einzelnen Datenblock passen. Die einzelnen Datensätze wurde auf mehrere Datenblöcke verteilt (die Datenblöcke bilden eine Kette, engl. "chain"). Da die Datensätze tatsächlich zu lang für einen Block sind, lässt sich dieses chaining nicht vermeiden.

Migrated Rows gehören in einen bestimmten Datenblock, haben aber dort keinen Platz mehr. Die einzelnen Datensätze liegen nun in anderen Datenblöcken. Dies geschieht, wenn Datensätze durch UPDATE Statements verlängert werden.

In beiden Fällen braucht es nun *mehr Festplattenzugriffe*, um die Datensätze zu lesen: Die Input/Output (IO) Performance sinkt.

Nach einem ANALYZE TABLE findet man in der Tabelle DBA_TABLES die Spalte CHAIN_CNT. Diese Spalte liefert die Anzahl von "chained rows" und "migratet rows".

- Falls der Anteil an migrierten Datensätzen sehr hoch ist, sollte man die Daten der Tabelle exportieren, die Tabelle mit grösserem PCTFREE anlegen und die Daten der Tabelle wieder importieren. Falls auf der Datenbank genügend Platz vorhanden ist, kann die auch in einer temporären Tabelle zwischenlagern. Mehr freier Platz pro Block verhindert Migration bei Updates, wenn die Datensätze länger werden.

- Falls der Anteil an verketteten Datensätzen hoch ist, sollte man die Daten zwischenlagern und die Tabelle mit einem kleineren PCTFREE neu an-

legen. Geringerer PCTFREE bedeutet, dass mehr Datensätze doch noch in einen Block passen könnten. Ob dies signifikant viele Datensätze sind, hängt von der Grössenverteilung ab.

```
SQL> select blocks, chain_cnt from user_tables
     where table_name = 'TEXTE';

    BLOCKS  CHAIN_CNT
---------- ----------
      2008          0

SQL> update texte set text = 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'
     where rownum < 10000;

9999 rows updated.

SQL> commit;

Commit complete.

SQL> analyze table texte compute statistics;

Table analyzed.

SQL> select blocks, chain_cnt from user_tables
     where table_name = 'TEXTE';

    BLOCKS  CHAIN_CNT
---------- ----------
      2409      15001
```

Mögliche Prüfungsfragen

- Wie entstehen "chained rows" und "migrated rows"?
- Zwischen PCTFREE und PCTUSED muss eine gewisse Menge Platz gelassen werden. Wie gross muss dieser Platz mindestens sein? Was wären die Folgen, wenn man dies vergisst?

2.4 Die Oracle Architektur Komponenten

Bei Oracle 8.1.5 unter Linux findet man diese Prozesse ganz einfach. Die SID der Datenbank ist in diesem Fall PERF.

```
/home/alex/texte/oracle $ ps u -U oracle
USER     PID %CPU %MEM  START   COMMAND
oracle   361  0.0  4.6  16:43   /opt/oracle/product/8.1.5/bin/tnslsnr LI
oracle   676  0.0  4.2  18:38   ora_pmon_PERF
oracle   678  0.0  4.3  18:38   ora_dbw0_PERF
oracle   680  0.0  4.1  18:38   ora_lgwr_PERF
```

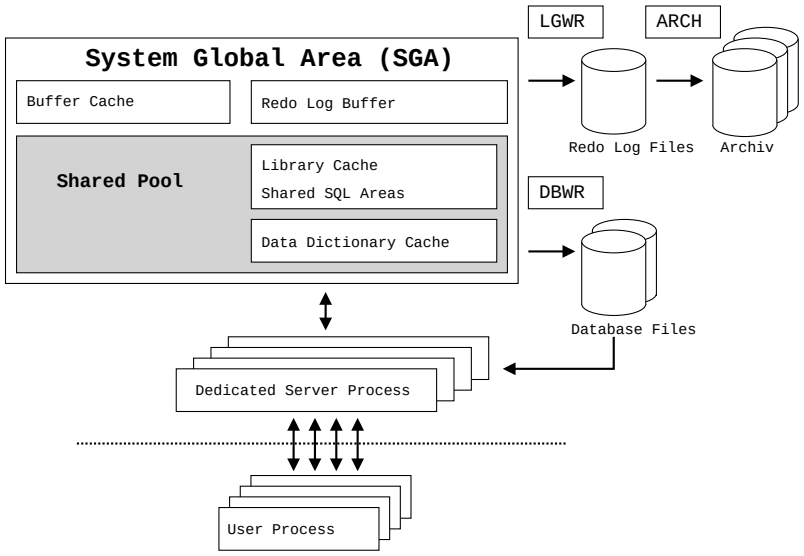

Abbildung 2.4: Oracle Architektur Komponenten

```
oracle   682  0.0  3.8  18:38   ora_ckpt_PERF
oracle   684  0.0  7.6  18:38   ora_smon_PERF
oracle   686  0.0  5.3  18:38   ora_reco_PERF
oracle   733  0.5  8.6  18:49   oraclePERF (DESCRIPTION=(LOCAL=no)(ADDRE
oracle   737  0.6  5.5  18:50   oraclePERF (DESCRIPTION=(LOCAL=no)(ADDRE
```

Im Moment sind zwei Verbindungen zur Datenbank offen. Die letzten beiden Einträge sind die Dedicated Server Prozesse. Die PERF Datenbank hat offensichtlich keinen ARCH Prozess. Die PMON, SMON, CHKPT und RECO Prozesse haben keinen regulierbaren Einfluss auf die Performance, so dass ich an dieser Stelle auf das Handbuch verweise. Siehe *Oracle7 Server Concepts*, Kapitel 9, Memory Structures and Processes.

User Processes Für jeden Benutzer, der sich auf der Datenbank anmeldet, existiert irgendwo im Netzwerk ein User Prozess. *Dieser User Prozess übernimmt die Kommunikation mit der Datenbank*, nimmt SQL Statements entgegen, übermittelt diese an die Datenbank, nimmt die Daten entgegen, und stellt diese der Applikation zu Verfügung. Die Kommunikation mit der Datenbank erfolgt oft über SQL*Net, ein Protokoll von Oracle.

- ODBC ist eine Alternative von Microsoft, welche von sehr vielen Datenbankherstellern unterstützt wird. ODBC ist deutlich langsamer als SQL*Net, wenn es um Oracle Datenbanken geht.
- JDBC wird von Java Programmen verwendet. JDBC wird auch von Oracle gefördert, da sich Oracle mit Oracle8 neu als Internet und Java Plattform etablieren will.

Dedicated Server Process *Für jeden Benutzer wird ein Dedicated Server Process gestartet.* Der Server Prozess nimmt die SQL Statements der Be-

nutzer entgegen und übernimmt das Holen der Daten und das Ausführen der SQL Statements.

Shared Pool Im Shared Pool liegen die Daten, welche der Server wiederverwenden kann. *Je grösser der Shared Pool ist, umso besser wird die Performance.* Der Shared Pool muss allerdings klein genug sein, um nicht auf die Festplatte ausgelagert zu werden.

Library Cache / Shared SQL Areas Im Library Cache befinden sich die Shared SQL Areas. Gibt es eine Shared SQL Area für ein identisches Statement im Shared Pool, so werden für das Statement die bestehende Shared SQL Area verwendet. Dies sind vor allem *der Parse Tree und der Execution Plan.*

Data Dictionary Cache *Um das Statement zu parsen und den Execution Plan aufzustellen werden die Informationen des Data Dictionaries verwendet.* Diese sind auch in einem eigenen Cache zu finden.

Buffer Cache *Der Server Prozess versucht, die benötigten Datenblöcke aus dem Buffer Cache zu lesen.* Im Buffer Cache stehen die bisher gelesenen Datenblöcke in einer Least Recently Used (LRU) Liste. Die Datenblöcke können Daten enthalten, die noch nicht in die Database Files geschrieben wurden. Man nennt diese Datenblöcke *dirty.*

Database Files *Falls der Server Process die Daten nicht im Buffer Cache gefunden hat, werden die Daten von den Database Files geholt* und in den Buffer Cache geschrieben. Falls es im Buffer Cache keinen Platz mehr gibt, werden zuerst die Blöcke durch den DBWR Prozess in die Database Files geschrieben und durch die neuen Blöcke ersetzt.

DBWR Der Database Writer Prozess schreibt die Daten aus dem Buffer Cache in die Database Files zurück. Dirty Blocks und lange nicht benutzte Blöcke werden zuerst zurück in die Database Files geschrieben. Alle drei Sekunden und an Checkpoints werden auch Daten in die Database Files geschrieben.

Redo Log Buffer Parallel zu der Änderung der Datenblöcke im Buffer Cache *werden alle Änderungsanweisungen im Redo Log Buffer gespeichert.* Der Redo Log Buffer enthält keine richtigen Datenblöcke sondern nur noch Änderungsanweisungen.

LGWR *Der Log Writer Prozess schreibt den Redo Log Buffer auf die Festplatte.* Alle drei Sekunden und sobald ein COMMIT erfolgt, werden die Daten in die Redo Log Files geschrieben.

Redo Log Files Die Redo Log Files enthalten einen kontinuierlichen Strom von Änderungsanweisungen. Wenn ein User eine Transaktion mit COMMIT abschliesst, werden die Redo Log Files geschrieben. *Das erfolgreiche Schreiben des Redo Log File Eintrages definiert ein COMMIT.* Wenn man eine Kopie alter Datenbank Dateien hat, kann man diese mit der Hilfe von Redo Log Files auf den neuesten Stand bringen. Dies geschieht beim Hochfahren des Servers automatisch.

ARCH Falls automatische Archivierung aktiviert ist, *wird der Archiver Prozess die Redo Log Files nach und nach auf weitere Datenträger kopieren.* Meist bedeutet dies, dass die Redo Log Files auf Tape gesichert werden.

Archiv Das Archiv besteht aus Tapes, entfernbaren Festplatten oder anderen Backup Lösungen. Das Archiv enthält die Redo Log Files. Zusammen mit (älteren) Backups der Datenbank Dateien können die Daten bis zur allerletzten abgeschlossenen Transaktion wieder hergestellt werden.

Auf dem Diagramm wurden *folgende Punkte ausgelassen*: Dnnn (Dispatcher Processes), Shared Server Processes, PGA (Program Global Areas), Software Code Areas, Sort Areas, Control Files, RECO (Recoverer), PMON (Process Monitor), SMON (System Monitor), CKPT (Checkpoint), LCKn (Lock Prozesse), SNP (Snapshot Refresh Prozesse). All diesen Punkten ist gemeinsam, dass sie keinen direkten Einfluss auf die Datenbank Performance haben. Details zu den einzelnen Punkten findet man in schön übersichtlicher Form im Handbuch *Oracle7 Server Concepts*, Kapitel 9, Memory Structures and Processes.

Oft spricht man im Zusammenhang mit Speicherbedarf von einer *Speicherhierarchie.*

- *Primärspeicher* ist der CPU direkt zugänglich. Es handelt sich um den Hauptspeicher des Rechners. Für die Daten in einer Oracle Datenbank ist dies der *Buffer Cache.*
- *Sekundärspeicher* ist der CPU nur indirekt zugänglich. Es handelt sich um Festplatten, CD-ROMs, etc. Daten auf diesen Medien müssen in den Primärspeicher kopiert werden, bevor sie verwendet werden können. Dies ist zwar um einiges langsamer, dafür ist das Medium viel billiger, so dass entsprechend mehr Platz zur Verfügung steht. Für die Daten in einer Oracle Datenbank ist sind dies die *Database Files.*
- *Tertiärspeicher* ist der CPU nicht zugänglich. Es handelt sich hierbei um offline Medien wie Magnetbänder. Der Zugriff auf diese Daten ist besonders langsam, da die Bänder erst online gebracht werden müssen, sei es durch einen Operator oder ein tape jukebox. Da das Medium allerdings noch viel billiger ist, steht entsprechend mehr Platz zur Verfügung. Für die Daten in einer Oracle Datenbank ist dies das Archiv.

Mögliche Prüfungsfragen

- Welche Komponenten der Oracle7 Architektur werden von den Datenbank Benutzern gemeinsam genutzt? Welchen Vorteil bietet das?
- Welche Komponenten der Oracle7 Architektur ermöglichen es, Daten zu ändern, ohne dass Oracle die geänderten Daten sofort in die Datenbank Dateien schreiben muss?
- Welche Komponenten der Oracle7 Architektur ermöglichen es, nach einem Crash die Daten bis zum letzten Commit wieder herzustellen?
- Welche Komponenten der Oracle7 Architektur ermöglichen es, Backups der Datenbank zu erstellen, ohne jedes Mal die gesamten Datenbank Dateien zu sichern?

- Welche Komponenten der Oracle7 Architektur ermöglichen es, die Daten auf einen Stand zu bringen, der an einem beliebigen Zeitpunkt in der Vergangenheit aktuell war? (Man nennt dies auch *point in time recovery*.)
- Welche Komponenten der Oracle7 Architektur limitieren die Performance?
- Welche Komponente der Oracle7 Architektur lässt sich bei einer schon bestehenden Datenbank am leichtesten manipulieren, um die Performance der Datenbank ganz allgemein zu steigern? Erklären sie, warum sich die Performance hiermit nicht beliebig steigern lässt.
- Was ist der Unterschied zwischen Redo Log Dateien und Datenbank Dateien? Warum werden beide benötigt?
- Welche Kriterien muss jeder Datenbank Architektur erfüllen?

Kapitel 3

Design

In diesem Teil beschäftigen wir uns mit ER Diagrammen: Wie sollen Tabellen erstellt werden, und wie sollen diese indexiert werden?

3.1 Normalisieren

Um zu einem normalisierten ER Diagramm zu kommen, gehen sie wie folgt vor:

1. Identifizieren sie die *Entitäten*. Entitäten müssen für das System (und somit für die Bedürfnisse des Kunden) *relevant* sein. Alle Entitäten müssen *unabhängig* voneinander existieren können.
2. Identifizieren sie den *Primärschlüssel* (primary key) der einzelnen Entitäten. Der Primärschlüssel identifiziert jeden Datensatz *eindeutig* (unique).
3. Identifizieren sie die *Beziehungen* (relations) zwischen den Entitäten. Bestimmen sie die Kardinalität dieser Beziehungen.
4. Identifizieren sie die *Attribute* der Entitäten. Oft tauchen hier neue Entitäten auf, so dass man hier wieder von vorne beginnt.

Mit diesem iterativen Prozess sollten sie bald ein normalisiertes ER Diagramm erhalten. Beachte sie, dass sie für M:M Beziehungen Mapping Tabellen erstellen müssen.

Hier ein ER Diagramm des SCOTT/TIGER Schemas nach Ausführen des utlsampl.sql Skriptes.

Mögliche Prüfungsfragen

- Nennen sie kurz die Vorteile einer Normalisierung.
- Beschreiben sie kurz die Notation für ER Diagramme.
- Beschreiben sie kurz den Normalisierungsprozess.
- Wie bilden sie M:M Beziehungen in einem ER Diagramm ab? Zeichnen sie das ER Diagramm für folgenden Sachverhalt: Ein Mitarbeiter kann

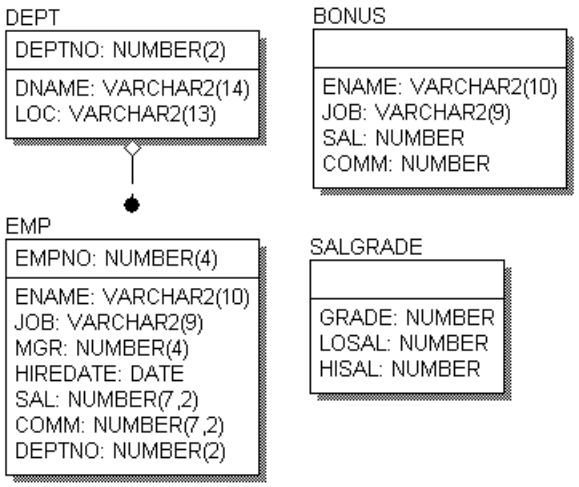

Abbildung 3.1: Beispiel für ein ER Diagramm

in mehreren Projekten arbeiten, in einem Projekt können mehrere Mitarbeiter arbeiten. Ein Mitarbeiter hat mindestens einen Vornamen (String), einen Nachnamen (String) und einen Titel (String). Jedes Projekt hat mindestens eine Projektnummer (Zahl), und einen Projektnamen (String).

3.2 Denormalisieren

Denormalisierung bedeutet immer, Daten redundant abzulegen. Werden die richtigen Daten redundant abgelegt, kann schneller auf die Daten zugegriffen werden. Redundante Daten bedeuten immer *neue Fehlerquellen, mehr Speicherplatz Bedarf und höherer Aufwand* bei der Datenpflege.

Zuerst ein Beispiel:

- Legen sie zwei einfache Tabellen an. In der ersten Tabelle werden Projekte gepflegt, in der zweiten Tabelle werden die Statusänderungen protokolliert. Damit ist klar, dass in derjenige Eintrag mit dem höchsten Datum den aktuellen Status reflektiert. Schreiben sie ein Select Statement, welches alle Projekte mit dem aktuellen Status anzeigt. Wir werden eine Änderung an der Projekt Tabelle durchführen, welche das Lesen der aktuellen Daten beschleunigt, das Pflegen der Daten allerdings verlangsamt.

Es folgen nun ein paar Beispiele; im Buch *Oracle Performance Tuning* finden sie weitere Beispiel, auf die ich allerdings nicht weiter eingehen will.

Aktuelle Version aus einer History doppelt pflegen Dies bietet sich an, wenn in einer abhängigen Tabelle Informationen über die Vergangenheit geführt werden. Da im Normalfall nur der aktuelle Wert interessiert, kann man diesen in der übergeordneten Tabelle führen, statt diesen jedesmal neu zu bestimmen.

Gewisse Konstanten verbatim aufführen Das Paradigma der vollständigen Normalisierung führt dazu, dass auch die einfachsten Attribute als Entitäten behandelt werden: Geschlecht (M, F), Gültigkeit (0, 1), etc. In diesen Fällen wird man diese Konstanten nicht in eigene Tabellen legen sondern direkt in die SQL Statements schreiben. Dies natürlich nur, weil man genau weiss, dass die Anzahl der Zustände fix ist.

Berechnete Werte mitführen Falls für bestimmte Übersichten immer dieselben Summen berechnet werden, kann man eine Kopie dieser Summen auch mitführen. Bugs, welche die Datenkonsistenz allerdings beeinträchtigen sind schwer zu finden und führen bei Kunden zu Vertrauensverlust und reduzierter Akzeptanz.

System-generierte Sequenzen als Primärschlüssel In relationalen Datenbanken muss der Primärschlüssel den Datensatz eindeutig identifizieren. In einer Tabelle mit Begriffen (z. B. Wörter der deutschen Sprache) ist der Begriff selber eindeutig. Trotzdem wird man den Begriff nicht zum Primärschlüssel machen, weil sonst Kopien der Begriffe auch in allen anderen Tabellen auftauchen. Das wäre Platzverschwendung. Wir werden bei den Indexen nochmals darauf zu sprechen kommen. Im Falle unserer TEXTE Tabelle werden wir also die ID Spalte in anderen Tabellen referenzieren und nicht die TEXT Spalte.

Stammdatentabellen kombinieren Bei Stammdaten gibt es oft nur wenige Einträge (Liste der Fehlerorte, Fehlerursachen, Apparatefamilien). Da jede Tabelle und jeder Index aber mindestens einen Block benötigt, wäre das eine Platzverschwendung. Deswegen soll man die Stammdaten in Gruppen zusammenfassen und alle in einer einzigen Tabelle ablegen. Beispiel: Tabelle CUST_CODE mit CODE_ID, CODE_TYPE, TEXT_ID und einer Tabelle TEXT mit TEXT_ID, LANGUAGE_ID, TEXT. In diesen Tabellen können Fehlerorte, Fehlerursachen und Apparatefamilien auf Deutsch und auf Englisch abgelegt werden.

Zusammenfassende Tabellen erstellen Oft sind werden für Reports nicht die aktuellsten Daten benötigt. Aus den aktuellen Daten kann über Nacht eine zusammenfassende Tabelle gefüllt werden, welche die Daten speziell für gewisse Reports zusammenfasst. Diese zusammenfassende Tabelle dient als kleines Datawarehouse.

Mögliche Prüfungsfragen

- Was ist ein Datawarehouse? Was bedeutet dies für die Datenbank?
- Was ist Online Transaction Processing (OLTP)? Was bedeutet dies für die Datenbank?
- Was ist ein Decision Support System? Was bedeutet dies für die Datenbank?
- Warum ist die Akzeptanz so wichtig?

- Begründen sie kurz die Vorteile und Nachteile einer Denormalisierung. Beschreiben sie drei Strategien für eine Steigerung der Performance mittels Denormalisierung. Für jede Strategie sollten sie die Voraussetzungen und mögliche Risiken für das Projekt aufführen.
- Ein Fertigungsauftrag hat eine Auftragsnummer (eine Zahl), eine Positionsnummer (eine Zahl), sowie eine SAP Fertigungsnummer (alphanumerisch, max. Länge 12), eine Stückzahl, einen aktuellen Status (eine Zahl), sowie eine Historie, aus der ersichtlich ist, welcher Datenbank Benutzer (alphanumerisch, max. Länge 12) zu welchem Zeitpunk (ein Datum) den Status geändert hat, sowie ein Bemerkungsfeld zur Statusänderung (alphanumerisch, max. Länge 200). Zeichnen sie zwei ER Diagramme: Das erste ER Diagramm soll vollständig normalisiert sein, das zweite ER Diagramm soll eine Änderung enthalten, welche die Performance der Applikation steigern soll. Begründen sie diese Änderung.
- Gegeben sind die beiden obigen ER Diagramme. Schreiben sie für beide Situationen ein SELECT Statement, welches alle Informationen über den aktuellen Zustand eines Fertigungsauftrages liefert. Begründen sie die bessere Performance.
- Gegeben ist obige Performance Steigerung. Beschreiben sie, wie Statusänderungen vorher gehandhabt wurden und wie sie jetzt gehandhabt werden.

3.3 Indexierung

Indexierung dient dazu, Daten schneller zu finden. Ohne Indexe muss ansonsten jedesmal die gesamte Tabelle durchsucht werden, selbst wenn nur ein einziger Datensatz benötigt wird.

Indices werden automatisch verwendet; die SQL Statements müssen nicht angepasst werden.

Der Index bezieht sich ein oder mehrere Attribute einer Tabelle. Bezieht sich der Index auf mehrere Attribute, spricht man von einem *Compound Index*. Oracle wird dann im Hintergrund alle indexierten Attribute miteinander verknüpfen und diesen Wert indexieren.

Grundsätzlich liefert ein Index für einen gesuchten Wert einen oder mehrere Datenpointer, welche den Ort auf der Festplatte bezeichnen, an dem sich der Datensatz mit dem entsprechenden Wert befindet.

Meistens kann man sich einen Index *wie eine sehr einfache Tabelle mit zwei Spalten vorstellen*: In der ersten Spalte stehen die indexierten Werte, in der zweiten Spalte steht der Datenpointer.

Dieses sehr einfache Modell beschreibt alle wesentlichen Eigenschaften:

1. *die indexierten Werte* für den Aufbau des Indexes,
2. *die Datenpointer*, welche auf den ursprünglichen Datensatz zeigen,
3. *im Index lässt sich ein Wert schneller finden*, da der Index kleiner als die ursprüngliche Tabelle ist, und
4. *der Index muss nachgeführt werden*, wenn sich die ursprüngliche Tabelle ändert

Der Rest des Kapitels befasst sich mit Methoden, das Finden der gesuchten Werte noch schneller zu machen. Alle Varianten *bieten Vorteile*, welche die Performance weiter steigern, und alle Varianten *haben Nachteile*, welche ihre Anwendung beschränken. Für eine optimale Performance einer Datenbank ist es wichtig, dass die richten Spalten indexiert werden, dass die richtige Index Variante gewählt wird, und dass die SQL Statements so geschrieben sind, dass die Indexe auch tatsächlich verwendet werden.

Indexe können auf alle mögliche Spalten (eindeutige und uneindeutige) angelegt werden. Die *80% Regel*: Alle Primary Keys und alle Foreign Keys werden indexiert. Weitere 5% Regel: Weitere Index nur anlegen, wenn sie tatsächlich gebraucht werden. Jeder Index verlangsamt Insert, Update und Delete Operationen.

Indices sollten für Spalten angelegt, welche oft *in WHERE Klauseln verwendet* werden. Spalten, die nur *innerhalb von Funktionen* in einer WHERE Klausel auftauchen, sollten nicht indexiert werden. In Oracle8 kann man hierfür einen Function Index verwenden.

Indices sollten für Spalten angelegt werden, welche *selektiv* sind. Spalten sind selektiv, wenn nur wenig andere Datensätze denselben Wert in derselben Spalte haben. Die Selektivität kann man berechnen, indem man die Anzahl Datensätze durch die Anzahl eindeutiger Index Werte teilt. Diese Information findet man im Data Dictionary (NUM_ROWS in USER_TABLES bzw. DISTINCT_KEYS in INDEX_STATS). In Oracle7 und Oracle8 Enterprise Edition kann man für nicht selektive Spalten einen Bitmap Index verwenden.

Indices sollten für Spalten angelegt werden, welche *oft für Joins verwendet* werden. Dies trifft für die meisten *Foreign Keys* zu. Dies hat zudem den Vorteil, dass die Referential Constraints bei UPDATE und DELETE Statements auf der Parent Table nicht zu einem Share Lock auf der gesamten Child Table führen.

Spalten, *deren Wert sich oft ändert*, sollten nicht indexiert werden, da bei jeder Änderung (UPDATE) die Indexe auch mit geändert werden müssen. Tabellen, bei denen oft Datensätze hinzugefügt oder gelöscht werden (INSERT oder DELETE), sollten nicht indexiert werden, da die Indexe auch jedesmal mit geändert werden müssen. Dies ist insbesondere bei OLTP Applikationen relevant.

Werden grosse Mengen an Daten neu eingefügt, sollten Indices gedropped werden und am Schluss wieder angelegt werden.

Für kleine Tabellen lohnt sich ein Index nicht, da zuerst ein Zugriff auf den Index und dann ein Zugriff auf die Tabelle erfolgt (mindestens zwei Datenblöcke). Wenn die Tabelle klein ist (wenige Blocks), dann ist der Zugriff am schnellsten, wenn die gesamte Tabelle auf ein Mal gelesen wird.

Mögliche Prüfungsfragen

- Gegeben ist ein ER Diagramm. Welche Indexe würden sie anlegen? Gehen sie davon aus, dass für die referentielle Integrität keine Indexe angelegt wurden.
- Warum indexiert man nicht jede Spalte in jeder Tabelle? Begründen sie die Antwort.
- Unter welchen Umständen kann sich ein Index negativ auf die Performance auswirken?

3.3.1 ROWID Beispiel für Oracle

Die Datenpointer, welche in einem Index verwendet werden, heissen im Oracle Jargon ROWID. Die ROWID verhält sich wie eine unsichtbare Spalte in jeder Oracle Tabelle. Für jede Zeile in der Tabelle bezeichnet die ROWID die physikalische Adresse der Zeile in der Datenbank.

Unter Oracle7 enthält die ROWID Block Address, Row, und File (BARF). Unter Oracle8 enthält die ROWID Segment, Datafile (pro Tablespace), Datablock (pro Datafile), und Row.

Hier ein Beispiel für Oracle8:

```
SQL> desc person;
 Name                                      Null?    Type
 ----------------------------------------- -------- -------------
 PERSONNR                                  NOT NULL NUMBER
 VORNAME                                            VARCHAR2(30)
 NACHNAME                                           VARCHAR2(30)

SQL> select personnr, substr(vorname||' '||nachname,1,20) name,
     rowid, substr(rowid,1,6) segment, substr(rowid,7,3) datafile,
     substr(rowid,10,6) block, substr(rowid,16,3) rownr
     from person;

PERSONNR NAME            ROWID              SEGMEN DAT BLOCK  ROW
-------- --------------- ------------------ ------ --- ------ ---
       1 Rudi Fischer    AAAAtLAABAAAFPaAAA AAAAtL AAB AAAFPa AAA
       2 Trudi Fischer   AAAAtLAABAAAFPaAAB AAAAtL AAB AAAFPa AAB
       3 Selene Meier    AAAAtLAABAAAFPaAAC AAAAtL AAB AAAFPa AAC
```

3.3.2 Die einfachsten Indexe

Die einfachsten Indexe bestehen aus *zwei Spalten*: In der zweiten Spalte stehen die Adressen aller Blocks, in denen Daten stehen, und in der ersten Spalte steht jeweils der Primärschlüssel des Datensatzes, mit dem der Block beginnt. Vorteil: Es gibt nur soviele Einträge im Index wie Blocks in der Tabelle. Nachteil: Die Daten in der Tabelle müssen nach dem Primärschlüssel sortiert sein. Dies ist beim Einfügen von neuen Datensätzen problematisch: Dies wird durch temporäre Bereiche gelöst, in denen neue Daten angelegt werden (overflow file oder overflow buffers). Gelöschte Daten werden als gelöscht markiert. Diese Markierungen werden zusammen mit den neuen Daten in regelmässigen Abständen in die Indexe und Datafiles eingearbeitet. Mehr Infos zu alten Indexierungsmöglichkeiten finden sie z. B. in *Fundamentals of Database Systems*.

Diese Index Variante wird von Oracle nicht eingesetzt.

3.3.3 Suchbäume

Suchbäume haben eine Ordnung `p`, so dass an jedem Knoten `p - 1` Werte und `p` Pointer auf weitere Knoten vorhanden sind. Die Pointer können leer sein.

Hier sehen sie einen Suchbaum für p = 3. Die Datensätze wurden in dieser Reihenfolge eingefügt: 5, 3 (links von 5), 13 (rechts von 5), 6 (zwischen 5 und

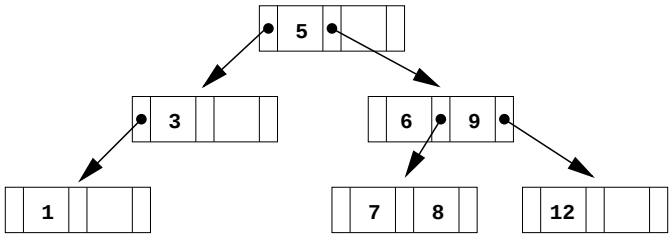

Abbildung 3.2: Suchbaum

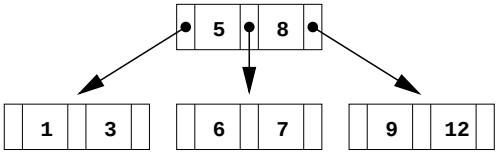

Abbildung 3.3: B-Tree

11), 1 (links von 3), 9 (rechts von 6), 7 (zwischen 6 und 9), 8 (rechts von 7), 12 (rechts von 12), 13 wurde gelöscht.

Suchbäume erlauben es, Werte in den Suchbaum aufzunehmen, ohne die zugrundeliegenden Daten zu sortieren.

Suchbäume werden nicht automatisch balanciert. Im obigen Fall befinden sich die Datenpointer verteile auf Stufen 1 bis 3, dh. der Zugriff dauert unterschiedlich lang. Zudem führt das Löschen von Datensätzen zu fast leeren Knoten, so dass sich die Zugriffe unnötig verzögern.

Diese Index Variante wird von Oracle nicht eingesetzt.

Mögliche Prüfungsfragen

- Suchbäume haben gegenüber B-Trees einen grossen Nachteil: Sie werden nicht automatisch balanciert, wenn Datensätze eingefügt oder gelöscht werden. Zeichnen sie ein Diagramm für einen Suchbaum mit Ordnung p = 3 in den die Datensätze wie folgt (Reihenfolge ist wichtig!) eingefügt werden: 1, 2, 3, 4, 5, 6, 7, 8.

3.3.4 B-Trees

B-Trees sind grundsätzlich wie Suchbäume aufgebaut. Zusätzlich wird beim Einfügen und Löschen dafür gesorgt, dass der Baum *immer balanciert* ist. Dies ist immer dann aufwendig, wenn ein neuer Pointer in einen schon vollen Knoten eingefügt werden muss (overflow), oder wenn die Anzahl Pointer eines Knoten unter ein bestimmtes Minimum sinkt (underflow).

B-Trees haben eine Ordnung `p`, so dass an jedem Knoten höchstens `p` und mindestens `p/2` Pointer auf weitere Knoten vorhanden sind. Wird die effektive Anzahl Pointer mit `q` bezeichnet, so gibt es in einem Knoten genau `q - 1` Werte mit entsprechenden Datenpointer, dh. höchstens `p - 1`.

Wird ein Wert in einen Knoten *eingefügt*, der schon `p - 1` Werte enthält, so wird der Knoten aufgeteilt: Der mittlere Wert wandert hoch, der Rest wird auf die zwei neue Knoten aufgeteilt.

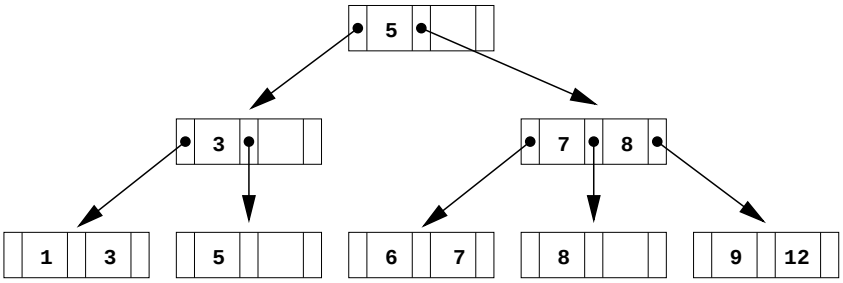

Abbildung 3.4: B$^+$-Tree

Im obigen Beispiel wurden die Werte in folgender Reihenfolge eingefügt: 8, 5 (links von 8), 1 (split, 5 wandert hoch), 7 (links von 8), 3 (rechts von 1), 12 (split, 8 wandert hoch), 9 (links von 12), 6 (links von 7).

Beim *Löschen* von Werten werden die Knoten mit den Nachbarknoten verschmolzen. Auch diese Veränderung kann bis zum obersten Knoten propagiert werden.

Der Füllgrad der Knoten muss irgendwo zwischen 50% und 100% liegen. Durch Analyse und Simulation konnte man zeigen, dass bei zufälligem Einfügen und Löschen von Werten die Knoten zu ungefähr 69% gefüllt sind.

B-Trees verhindern unbalancierte Suchbäume, so dass keine Werte ausserordentlich langsam zu finden sind.

Diese Index Variante wird von Oracle nicht eingesetzt. Die von Oracle verwendeten B*-Trees sind allerdings Spezialfälle von B-Trees, so dass man ab und zu oft ganz allgemein von B-Trees spricht.

3.3.5 B$^+$-Trees

B$^+$-Trees sind grundsätzlich wie B-Trees aufgebaut. In einem B$^+$-Tree befinden sich *alle Datenpointer bei den Blättern des Baumes*. Werte, welche in den Knoten des Baumes auftauchen, tauchen demzufolge mehrmals im Baum auf. Die Blätter des Baumes sind meistens doppelt miteinander verkettet.

Da in den Knoten der B$^+$-Trees keine Datenpointer auftauchen, können B$^+$-Tree Knoten *mehr Werte und Pointer als B-Tree Knoten* enthalten. Pro Block kann ein B$^+$-Tree also eine höhere Ordnung `p` haben. Demzufolge kann ein solcher B$^+$-Tree möglicherweise aus weniger Ebenen bestehen als ein B-Tree.

Im obigen Beispiel wurden die Werte in folgender Reihenfolge eingefügt: 8, 5 (links von 8), 1 (split, 5 wandert hoch), 7 (links von 8), 3 (split, 3 wandert hoch), 12 (split, 8 wandert hoch, propagiert, split, 5 wandert hoch), 9 (links von 12), 6 (split, 7 wandert hoch).

B$^+$-Trees erlauben es, alle Werte in konstanter Zeit zu finden. Das Verlinken der Blätter des Baumes erlaubt neben der Suche nach exakten Werten auch das Suchen nach Werten in einem bestimmten Bereich.

Diese Index Variante wird von Oracle nicht eingesetzt. Die von Oracle verwendeten B*-Trees sind allerdings nur Spezialfälle von B$^+$-Trees, so dass man oft genug ganz allgemein von B$^+$-Trees spricht.

3.3.6 B*-Trees

B*-Trees sind grundsätzlich wie B^+-Trees aufgebaut. In einem B*-Tree wird verlangt, dass ein Knoten nicht nur zu 50% voll ist, sondern 67% voll ist. Bei Oracle ist es sogar so, dass die Knoten zu 75% voll sind.

Dies sind die normalen Indexe, welche von Oracle verwendet werden. Wenn die Index Variante nicht angegeben ist, spricht man im allgemeinen von B*-Trees.

3.3.7 Function Indexes

Spalten, die nur *innerhalb von Funktionen* in einer WHERE Klausel auftauchen, sollten mit Oracle7 nicht indexiert werden, da in diesem Fall ein normaler Index nicht verwendet wird. In Oracle8 kann man hierfür einen Function Index verwenden.

Function Indexes bieten sich nur an, *wenn immer dieselbe Funktion* auf eine Spalte angewendet wird. Der Function Index indexiert nicht Werte sondern das jeweilige Funktionsergebnis.

Nur der *Cost Based Optimizer* (CBO) kann Function Indexes verwenden.

3.3.8 Bitmap Indexes

Normale Indexe sollten nur für Spalten angelegt werden, welche *selektiv* sind, da der Index sonst sehr gross wird und trotzdem sehr wenig bringt.

Ein Bitmap Index eignet sich *für nicht-selektive Spalten* (man spricht auch von niedriger Kardinalität). Dies ist dann der Fall, wenn alle Werte mehr als hundert Mal in den indexierten Spalten auftauchen.

Nr	Name	Geschlecht
1	Fischer	M
2	Müller	M
3	Wehner	F

Entsprechender Bitmap Index:

Geschlecht=‘F’	Geschlecht=‘M’
0	1
0	1
1	0

Bitmap Indexe enthalten auch NULL Werte. Um dasselbe Resultat wie normale Indexe zu garantieren, wird Oracle jeweils die Teilmenge mit NULL Werten vom Resultat wieder abziehen. Um dies zu vermeiden, sollten Spalten, welche keine NULL Werte enthalten, als NOT NULL Spalten deklariert werden.

Im folgenden Beispiel könnte hiermit die zweite minus Operation von Oracle weggelassen werden, wenn die Spalte PARTY als *NOT NULL deklariert* worden wäre:

```
SQL> SELECT COUNT(*) FROM people
     WHERE state='CA' and party !='D';

...
```

```
Execution Plan
----------------------------------------------------------

   SELECT STATEMENT
  SORT                    AGGREGATE
    BITMAP CONVERSION     COUNT
      BITMAP MINUS
        BITMAP MINUS
          BITMAP INDEX    SINGLE VALUE    STATE_BM
          BITMAP INDEX    SINGLE VALUE    PARTY_BM
        BITMAP INDEX      SINGLE VALUE    PARTY_BM
```

Die Platzersparniss mit Bitmap Indexe wird noch wichtiger, wenn die Alternative eine Reihe zusammengesetzter Index ist. Beispiel: Die Tabelle PERSON enthält die Resultate einer Statistik zum Wahlverhalten. Die Tabelle hat unter anderem die Attribute GENDER, PARTY, AGE_CATEGORY, INCOME_CATEGORY, ZIP_CODE, LANGUAGE. Bei den Abfragen können alle oder nur einzelne dieser Attribute eingeschränkt werden, das bedeutet entweder normale Indexe auf jede einzelne Spalte, was zu vielen Sortiervorgängen führt, wenn die Schnittmenge der Teil-Resultat ermittelt werden, oder eine grosse Anzahl kombinierter Indexe für die häufigsten Permutationen von Attributen, was wiederum sehr viel Platz verbraucht und das Ändern der Daten verlangsamt. Dieses Problem kann elegant gelöst werden, wenn für jede Spalte ein Bitmap Index angelegt wird.

Bitmap Indexe brauchen in diesen Fällen weniger Platz als B-Trees. Werden Daten mittels Update Statements geändert, können Einträge in einem Index gesperrt werden. Handelt es sich um einen B-Tree Index, so wird nur ein Eintrag gesperrt. Handelt es sich allerdings um einen Bitmap Index, können viel mehr Einträge gesperrt werden. Deswegen sind *Bitmap Indexe nicht für OLTP Applikation geeignet.*

Der Rule Based Optimizer (RBO) berücksichtigt keine Bitmap Indexe. *Nur der Cost Based Optimizer (CBO) berücksichtigt Bitmap Indexe.*

In Oracle8 sind Bitmap Indexe nur in der Enterprise Edition möglich; die Standard Edition enthält keine Bitmap Indexe mehr.

3.4 Clusters

Die bisher besprochenen Indexe (B-Trees, Function, Bitmap) können für bestehende Tabellen angelegt werden. Es gibt aber auch die Möglichkeit, die *Daten in den Tabellen für bestimmte Zugriffsarten optimiert abzulegen.*

Tabellen können zu einem *Cluster* zusammengefasst werden. Hierfür muss ein *Cluster Key* bestimmt werden.

Normalerweise erfolgt der Zugriff auf die Datensätze eines Clusters über einen Cluster Index. Handelt es sich jedoch um einen Hash Cluster, so wird eine Hashing Funktion verwendet, um die Datensätze zu finden.

3.4.1 Cluster Index

Alle Datenzeilen mit demselben Cluster Key werden physikalisch zusammen in denselben Blocks gespeichert. Mehrere Tabellen können kombiniert werden

(z. B. EMP und DEPT Tabelle mit Cluster Key DEPTNO). Der Cluster Key wird pro Cluster nur einmal gespeichert. Vorteile: Wenn alle Datensätze mit einem bestimmten Cluster Key gelesen werden, geht dies besonders schnell. Wenn der Cluster Key besonders viel Platz in Anspruch nimmt, wäre es zudem theoretisch möglich, so Platz zu sparen.

Clusters bieten sich an, wenn die Tabellen des Clusters oft in Joins zusammen verwendet werden. Dies ist insbesondere bei Master-Detail Beziehungen der Fall. Clusters sollten vermieden werden, wenn diese Joins nur ab und zu gemacht werden.

Nachteile: Werden neue Daten für einen bestehenden Cluster eingefügt, und es ist kein Platz mehr frei, werden die Datensätze migriert. Wird der Cluster Key mittels update geändert, müssen die Datensätze physikalisch verschoben werden. Dies führt möglicherweise zu noch *mehr Migration.*

Clusters sollten vermieden werden, wenn der Cluster Key oft geändert wird, da dies zu Migration von Datensätzen führt.

Clusters sollten vermieden werden, wenn nur von einigen der Cluster Tabellen ein Full Table Scan gemacht werden soll. In so einem Fall wird Oracle mehr Blocks als normalerweise lesen müssen. In diesem Fall kann man allerdings die verbleibenden Tabellen trotzdem noch zu einem Cluster zusammenfassen. Falls für eine Master-Detail Beziehung also Full Table Scans auf die Master oder die Detail Tabellen vorgesehen sind, dann kann man *die Detail Tabellen immer noch zu einem Cluster zusammenfassen.* In diesem Sinne kann ein Cluster *auch für einzelne Tabellen sinnvoll* sein.

Clusters sollten vermieden werden, wenn die Daten aller Tabellen für den Cluster Key mehr als ein oder zwei Blocks benötigen. In diesen Fällen muss Oracle für den Zugriff auf eine Datenzeile möglicherweise mehr als einen Block für den entsprechenden Cluster Key lesen, da ja alle Datensätze der verschiedenen Cluster Tabellen in denselben Blocks gespeichert sind.

3.4.2 Hash Clusters

Hash Clusters ähneln normalen Clusters: Mehrere Tabellen können einen Cluster bilden; *alle Datensätze mit demselben Cluster Key werden zusammen abgespeichert.*

Hash Clusters haben *dieselben Nachteil wie normale Clusters*: Änderung des Keys führt zur Migration von Datensätzen, Full Tabel Scans über einzelne Tabellen des Clusters werden aufwändiger, der Zugriff auf Datensätze wird aufwändiger, falls für alle Datensätze mit demselben Key mehr als ein oder zwei Blocks benötigt werden.

Hash Clusters bieten allerdings *bedeutende Vorteile*:

Neu ist der der Einsatz einer Hashing Funktion. *Die Hashing Funktion liefert für einen bestimmten Cluster Key die Adresse des Blocks, der diesem Key zugeteilt wurde.* Diese Zuteilung geschieht beim Erstellen des Hash Clusters.

Im günstigsten Fall kann ein Datensatz so mit nur einem Zugriff gelesen oder geschrieben werden.

Hash Clusters bieten sich an, wenn abgeschätzt werden kann, *wieviel Platz alle Datensätze mit einem bestimmten Cluster Key* benötigen werden. Genauso wie bei einem normalen Cluster sollte dies weniger als ein Block sein. Zusätzlich sollte aber bekannt sein, wieviele verschiedene Cluster Keys pro Block gespeichert werden können.

Leider wird der grosse Vorteil eines Hash Clusters durch einige *Nachteile* erkauft: *Die ungefähre Anzahl Hash Keys und die ungefähre Anzahl Datensätze pro Hash Key muss bekannt sein. Die Hash Keys müssen ungefähr gleichverteilt sein.*

Sind die Hash Keys nicht gleichverteilt, so werden die Datenblöcken, welche die Daten für einen relativ häufigen Hash Key enthalten, schnell voll sein. Die restlichen Daten werden in *overflow blocks* gespeichert. Die Hashing Funktion zeigt immer noch auf den ursprünglichen Block, den sog. *root block*. Die overflow blocks werden mittels *chaining* an den root block angehängt.

Hash Clusters sollten vermieden werden, wenn die Tabelle ständig wächst und der *Hash Cluster nicht hin und wieder grösser neu angelegt* werden kann.

Hash Clusters sollten vermieden werden, wenn *viele Full Table Scans* gemacht werden und *viel Platz für zukünftige Datensätze reserviert* wurde. In diesem Fall werden viele Blocks unnötigerweise gelesen. Bei einem normalen Cluster war dies nur problematisch, falls man für einen Teil der Tabellen im Cluster einen Full Table Scan machen wollte. Bei Hash Clustern ist dies ein Problem, selbst wenn es nur eine Tabelle im Hash Cluster gibt.

3.4.3 Index-organisierte Tabellen

Falls Oracle alle benötigten Daten beim Zugriff auf einen normalen B*-Tree Index findet, greift Oracle nicht auf die dem Index zugrundeliegende Tabelle zurück. Dies führt dazu, dass es sich unter bestimmten Umständen lohnt, weitere Spalten in einen Index aufzunehmen. Möglicherweise müssen bestimmte SQL Statements dann nicht mehr auf die Tabelle zugreifen und können nur den Index verwenden.

Diese Idee wurde von Oracle weiter entwickelt: Mit Oracle8 ist es möglich, Daten in Index-organisierten Tabellen abzulegen. *Index-organisierte Tabellen sind weder reine Indexe noch reine Tabellen.* Grundsätzlich handelt es sich um einen B*-Tree Index, auf dessen untersten Ebene sich nicht Datenpointer (dh. ROWIDs) befinden, sondern alle restlichen Daten. Die ROWID gibt es nicht mehr.

Werden *Sekundärindexe* angelegt, dh. traditionelle B*-Tree Indexe, dann liefern diese für einen gesuchten Wert eine oder mehrere ROWIDs, mit denen die Daten dann geholt werden können. Wenn es sich Datensätze in einer Index-organisierten Tabelle handelt, wird eine *logische ROWID* verwendet. Die logische ROWID basiert auf dem Primärschlüssel und enthält einen *Schätzwert* für die physische Adresse des Datensatzes. Falls dieser Schätzwert nicht aktuell ist, wird der Zugriff verlangsamt, da für jeden Datensatz der normale Primärschlüssel Index durchsucht werden muss.

Der Grund für das Fehlen einer festen physischen Adresse liegt im Aufbau der Index-organisierten Tabellen: Beim Einfügen neuer Daten kann es passieren, dass der *B*-Tree reorganisiert* werden muss. Die Datensätze werden verschoben. Sekundärindexe, welche derartige Schätzwerte enthalten, müssen regelmässig neu aufgebaut werden, um die Schätzwerte zu korrigieren.

Da die B*-Tree Indexe eigentlich sehr kompakt sind, in einer Index-organisierten Tabelle aber auf unterster Ebene aber viel mehr Daten gespeichert werden können, kann es passieren, dass der *Index nicht sehr kompakt* ist. Um dies zu verhindern, kann man beim Erstellen der Index-organisierten Tabelle ein OVERFLOW angegeben werden. Dies beinhaltet zwei Dinge:

1. PCTTHRESHOLD Parameter: Falls der Platzverbrauch eines Datensatzes *grösser als der angegebene Prozentsatz eines Blocks* ist, werden die "überzähligen" nicht-Schlüsselattribute ausgelagert.

2. Overflow Tablespace: Dies bezeichnet den *Tablespace*, in den die "überzähligen" nicht-Schlüsselattribute ausgelagert werden.

Das Lesen derartiger aufgeteilten Datensätze ist natürlich *langsamer als vorher*. Im Gegenzug ist die Index-organisierten Tabelle aber kompakter.

Anwendungsgebiete

Index-organisierten Tabellen werden von Oracle für folgende Applikationen empfohlen:

1. Information Retrieval (IR) Systeme, wo grosse Datenbestände durchsucht werden müssen

2. Online Analytical Processing (OLAP) Systeme, wo sehr viele Daten zusammengefasst und aufbereitet werden

3. Geographic Information Systeme (GIS), wo mit räumlichen Daten gearbeitet wird

Diesen Anwendungen ist gemeinsam, dass sie *invertierte Indexe* verwenden.
Beispiel Volltextsuche:
Die Wörter und Dokumente werden in zwei Tabellen gespeichert:

```
Daten für Tabelle DOKUMENT

DOKUMENT_NR   FILENAME
  1           /proj/643128/mails/76
  2           /proj/7359/mails/947
  2           /proj/234/mails/9

Daten für Tabelle OCCURRENCES

DOKUMENT_NR   TOKEN       OCCURRENCES
  1           Aachen        5
  1           azurblau      3
. . .         . . .       . . .
  2           Azteken       2
. . .         . . .       . . .
```

Normalerweise würde man eine Master-Detail Beziehung zwischen Dokumenten und den Wörtern in den Dokumenten aufbauen. So kann man zu jedem Dokument die Wörter finden, welche sich darin befinden. Diese Information ist *in einem IR System nutzlos*. Ein *invertierter Index* führt für jedes Wort auf, in welchen Dokumenten es zu finden ist.

```
Index für Tabelle OCCURRENCES

TOKEN      DOKUMENT_NR   ROWID
Aachen       1           . . .
azurblau     1           . . .
Azteken      2           . . .
```

In diesem Index werden alle oder fast alle Informationen noch einmal wiederholt. Das ist *Platzverschwendung*. Im obigen Fall könnte man als ersten Tuning Schritt die Spalte OCCURRENCES natürlich in den Index aufnehmen, das würde den Zugriff mittels ROWID auf die Tabelle OCCURRENCES einsparen, die Platzverschwendung allerdings noch verstärken. Dies kann man sich in einer Index-organisierten Tabellen sparen.

```
Index organisierte Tabelle OCCURRENCES

TOKEN       DOKUMENT_NR   OCCURRENCES
Aachen        1             5
azurblau      1             3
Azteken       2             2
```

Allerdings ist es auch hier so, dass man weitere Informationen zum gefundenen Dokument wieder über einen Index in der DOKUMENT Tabelle nachschauen muss. Index-organisierte Tabellen können nicht in Clustern verwendet werden.

Mögliche Prüfungsfragen

- Was für einen Index würden sie anlegen, um Projekte zu indexieren? Der Primärschlüssel ist eine fortlaufende Sequenznummer. Erklären sie die Vorteile gegenüber zwei anderen Indexierungsvarianten.
- Was für einen Index würden sie anlegen, um die Aktivitäten zu einem Projekt zu indexieren? Jede Aktivität enthält eine Referenz auf ein Projekt. Was für eine Rolle spielt die Projektnummer in der Aktivitäten Tabelle: Fremdschlüssel, Teil des Primärschlüssels, oder etwas anderes? Erklären sie ihre Wahl.
- Was für einen Index würden sie anlegen, um die Stichwörter der Pendenzen in einem CRM System zu indexieren? Gehen sie davon aus, dass alle Stichworte zusammen in *einem Attribut pro Pendenz* auf der Datenbank gespeichert werden (z. B. “Orbit Stand, Anmeldung, Prospekte”). Dies bedeutet, dass der Benutzer Volltextsuchen auf dieser Spalte ausführen will. Begründen sie ihren Entscheid.
- Was für einen Index würden sie anlegen, um die Gültigkeit von Artikeln zu indexieren? Nehmen sie an, dass die Artikeldaten relativ stabil sind. Ein Artikel kann gültig oder ungültig sein. Erklären sie den Vorteile gegenüber zwei anderen Indexierungsvarianten.
- Was für einen Index würden sie anlegen, um die Liste der Investigators (Personen) für eine Clinical Trial (Klinische Versuche neuer Medikamente an Menschen) zu indexieren? Der Primärschlüssel für die Investigators ist eine fortlaufende Sequenznummer. Erklären sie die Vorteile gegenüber zwei anderen Indexierungsvarianten.
- Was für einen Index würden sie anlegen, um die Firmennamen einer Firmentabelle zu indexieren? Gehen sie davon aus, dass die Benutzer immer den Anfang der Firmennamen angeben werden (dh. keine ’%xxx’ wildcard

Suchen). Gehen sie zudem davon aus, dass Gross- und Kleinschreibung sowie Whitespace (Leerschläge, Tabulatoren, Zeilenumbrüche) bei der Suche keine Rolle spielen dürfen. Erklären sie, wie ihre Lösung diese Bedingungen erfüllt.

- Was für einen Index würden sie anlegen, um den Status von Produkten zu indexieren? Nehmen sie an, dass der Produktestamm nur selten geändert wird, aber sehr intensiv gelesen wird (z. B. im Rahmen einer Web Applikation). Nehmen sie an, dass es nur die Stati 'geplant', 'gültig', 'gesperrt' und 'nicht mehr lieferbar' gibt. Erklären sie den Vorteile gegenüber zwei anderen Indexierungsvarianten.
- Was für einen Index würden sie anlegen, um die Liste der gültigen Projektstati zu indexieren? Es gibt in dieser Tabelle wahrscheinlich um die 10 Projektstati in jeweils drei Sprachen. Achtung: Es geht nicht um das Attribut Projektstatus in der Tabelle Projekt, sondern um eine Stammdatentabelle, in der die gültigen Werte für die das Attribut aufgeführt sind. So kann man dem Benutzer der Applikation eine entsprechende Auswahl in der gewünschten Sprache bieten. Erklären sie den Vorteile gegenüber zwei anderen Indexierungsvarianten.
- Was für einen Index würden sie anlegen, um eine Preistabelle zu indexieren? In dieser Tabelle werden pro Zeitraum (GUELTIG_VON, GUELTIG_BIS), Region (REGION_NR) und Artikel (ARTIKEL_NR) der Preis (PREIS) und die Währung (WAEHRUNG_NR) abgespeichert. Gehen sie davon aus, dass dieses Sortiment laufend nachgepflegt wird. Erklären sie den Vorteile gegenüber zwei anderen Indexierungsvarianten.
- Was für Indexe würden sie anlegen, um die Zeiterfassung von Mitarbeitern zu indexieren? Diese Daten werden regelmässig verwendet, um Listen der geleisteten Arbeiten pro Mitarbeiter und Monat (Arbeitszeiterfassung) oder pro Projekt und Monat (Stundenrapport für Arbeit nach Aufwand) zu generieren. Erklären sie ihre Wahl.
- Was für Indexe würden sie anlegen, um die Resultate von Bodenproben zu indexieren? Gehen sie davon aus, dass diese Daten dazu verwendet werden, um die Bodenbeschaffenheit an gewissen Punkten auf einer Karte zusammenzustellen. Erklären sie ihre Wahl.
- Bei gewissen Such-Bedingungen kann ein normaler Index nicht verwendet werden. Geben sie drei Beispiele. Geben sie an, ob möglicherweise ein spezieller Index Typ eingesetzt werden könnte.
- Bei gewissen Such-Bedingungen kann weder ein normaler Index noch ein Function Index nicht verwendet werden. Geben sie ein Beispiel und erklären sie, warum der Function Index nicht verwendet wird.
- Geben sie drei mögliche Gründe an, ein Index nicht verwenden zu wollen.

3.5 Indexe und der Data Dictionary

Bei der Erstellung der Index kann man, wie bei den Tabellen, Storage Parameters angeben. Genau wie dort gelten die Tablespace Defaults, wenn man keine

Storage Parameters angibt. *Alle relevanten Informationen lassen sich aus dem Data Dictionary wieder auslesen.*

Gewisse Informationen im Data Dictionary werden nur nach Bedarf bereit gestellt. Dies geschieht mit dem Befehlen ANALYZE INDEX:

```
ANALYZE INDEX index_name COMPUTE STATISTICS;
```

Dies berechnet Statistik Daten, welche in den USER_INDEXES, ALL_INDEXES und DBA_INDEXES Views ersichtlich sind.

```
ANALYZE INDEX index_name VALIDATE STRUCTURE;
```

Dies berechnet Statistik Daten, welche in den INDEX_STATS und INDEX_HISTOGRAM Views ersichtlich sind.

Beispiel: Die Grösse eines Indexes

In diesem Beispiel wird ein Index auf die TEXT Spalte der TEXTE Tabelle angelegt. Da sich fast alle Daten der Tabelle in dieser Spalte befinden, und sich die indexierten Werte im Index selber dupliziert werden, muss der Index ähnlich gross wie die Tabelle sein.

```
SQL> desc texte;
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 ID                                                 NUMBER
 TEXT                                               VARCHAR2(30)

SQL> select blocks from user_tables where table_name = 'TEXTE';

    BLOCKS
----------
       448

SQL> create index aktext on texte(text);

Index created.

SQL> analyze index aktext compute statistics;

Index analyzed.

SQL> select substr(index_name,1,10) index_name,
            index_type, leaf_blocks, blevel
     from user_indexes
     where table_name = 'TEXTE';

INDEX_NAME INDEX_TYPE                   LEAF_BLOCKS     BLEVEL
---------- ---------------------------- ----------- ----------
AKTEXT     NORMAL                               506          2
```

Dasselbe lässt sich auch noch in Bytes statt Blocks ausdrücken, obwohl man die Daten auf etwas anderem Weg erhält:

```
SQL> connect system/manager@perf
Connected.
SQL> select p.value * u.blocks table_size
     from v$parameter p, all_tables u
     where p.name = 'db_block_size'
     and u.table_name = 'TEXTE';

TABLE_SIZE
----------
    917504

SQL> connect scott/tiger@perf
Connected.
SQL> analyze index aktext validate structure;

Index analyzed.

SQL> select used_space, btree_space from index_stats
     where name = 'AKTEXT';

USED_SPACE BTREE_SPACE
---------- -----------
    838764      950300
```

Neben den Informationen über den Index selber, findet man auch Informationen zu denüber die Index Spalten im Data Dictionary.

```
SQL> select substr(index_name,1,15) index_name,
            substr(column_name,1,15) column_name,
            column_position
     from user_ind_columns
     where table_name = 'TEXTE';

INDEX_NAME      COLUMN_NAME     COLUMN_POSITION
--------------- --------------- ---------------
AKTEXT          TEXT                          1
```

3.6 Beispiel für die Verwendung eines Indexes

Zuerst legen wir die Testdaten an:

```
drop table person;
create table person (
  personnr number primary key,
  vorname varchar2(30),
  nachname varchar2(30));
insert into person (personnr, vorname, nachname)
  values (1, 'Rudi', 'Fischer');
insert into person (personnr, vorname, nachname)
  values (2, 'Trudi', 'Fischer');
```

```
insert into person (personnr, vorname, nachname)
  values (3, 'Selene', 'Schneider');
commit;
```

Wie wir sehen, wurde *für den Primary Key ein automatischer Index* namens SYS_C00877 angelegt:

```
SQL> select index_name, table_name, column_name
     from user_ind_columns
     where table_name = 'PERSON';

INDEX_NAME                     TABLE_NAME
------------------------------ ------------------------------
COLUMN_NAME
--------------------------------------------------------------
SYS_C00877                     PERSON
PERSONNR
```

Die Verwendung des Indexes geschieht für den Benutzer allerdings transparent. Hier zwei Select Statements; das erste Statement verwendet den Index, das zweite Statement verwendet keinen Index:

```
SQL> select vorname, nachname from person
     where personnr = 1;

VORNAME                        NACHNAME
------------------------------ ------------------------------
Rudi                           Fischer

SQL> select vorname, nachname from person
     where nachname = 'Fischer';

VORNAME                        NACHNAME
------------------------------ ------------------------------
Rudi                           Fischer
Trudi                          Fischer
```

Ob ein Index verwendet wird oder nicht, wird im *Execution Plan* festgelegt. Oracle legt den Execution Plan auf Wunsch in der Tabelle PLAN_TABLE an. Diese Tabelle muss man als Entwickler zuerst anlegen. Hierfür verwendet man das utlxplan.sql Skript im Verzeichnis $ORACLE_HOME/rdbms/admin, oder man legt eine entsprechende Tabelle von Hand an.

Dannach kann man sich von SQL*Plus den Execution Plan für alle Statements anzeigen lassen. Hierfür muss man die AUTOTRACE Option anschalten:

```
SQL> set autotrace on
```

Der Execution Plan wird entweder vom Cost Based Optimizer (CBO) oder vom Rule Based Optimizer (RBO) erstellt. Da der CBO ein einigermassen aktuelles ANALYZE TABLE voraussetzt, *forcieren wir die Verwendung des RBO*:

```
SQL> alter session set optimizer_goal=rule;

Session altered.
```

Hier nun die beiden Select Statements noch einmal. Der AUTOTRACE Output wurde etwas gekürzt.

```
SQL> select vorname, nachname from person where personnr = 1;

VORNAME                        NACHNAME
------------------------------ --------------------------
Rudi                           Fischer

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=RULE
   1    0   TABLE ACCESS (BY INDEX ROWID) OF 'PERSON'
   2    1     INDEX (UNIQUE SCAN) OF 'SYS_C00877' (UNIQUE)

SQL> select vorname, nachname from person
     where nachname = 'Fischer';

VORNAME                        NACHNAME
------------------------------ --------------------------
Rudi                           Fischer
Trudi                          Fischer

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=RULE
   1    0   TABLE ACCESS (FULL) OF 'PERSON'
```

Kapitel 4

Applikation

Eine Applikation, welche eine relationale Datenbank verwendet, wird diese über SQL Statements ansprechen. Es gibt hier zwei Möglichkeiten, die Performance der Applikation zu verbessern:

1. Redesign: *Die Applikation wird umgeschrieben*, so dass andere Daten angezeigt werden. Beispielsweise werden in einer gewissen Liste nicht mehr alle Artikel sondern nur die gültigen Artikel angezeigt.
2. SQL Tuning: *Die SQL Statements werden überarbeitet*, so dass die selben Daten schneller von der Datenbank geholt werden.

4.1 Redesign

Das Umschreiben der Applikation ist kostspielig: Mit dem Kunden muss abgeklärt werden, welche Alternativen in Frage kommen (Analyse). Eine der Lösungen muss ausgewählt und ausformuliert werden (Design). Diese Entscheidung muss schriftlich festgehalten werden (Spezifikation). Die Lösung muss implementiert werden (Implementierung). Testprozeduren müssen nachgeführt werden und die Tests müssen erneut durchgeführt werden. Die Lösung muss installiert werden. Je nach Umständen muss der Kunde seine Arbeit während der Installation und den Systemintegrationstests unterbrechen.

Fazit: *Während der Spezifikation der Software muss man schon ein Auge auf die Performance werfen.* Wenn der Kunde Joins über zehn Tabellen wünscht, wenn der Kunde komplexe Berechnungen bei jeder Validierung von Datenfeldern wünscht, wenn der Kunde Freitextsuchen über grosse Datenbestände wünscht, dann muss der Lieferant auch *als Berater auftreten* und den Kunden auf die möglichen Performance Nachteile hinweisen, bzw. die Daten auf eine Art und Weise ablegen, welche die gewünschte Performance ermöglicht.

4.2 SQL Tuning

SQL Statements überarbeiten ist eine Aktivität, welche die Entwickler *immer wieder* machen müssen. Vom ersten Entwurf im Haus bis zur neuesten Problemmeldung vom Kunden, immer geht es um die Performance einzelner SQL Statements.

Je nach Applikation sind die Ziele für SQL Tuning verschieden:

OLTP Viele Transaktionen, Ressourcenverbrauch minimieren

Datawarehouse Maximaler Datendurchsatz, Parallele Verarbeitung

Natürlich profitieren Datawarehouse Applikationen auch von gut geschriebenen SQL Statements, welche den Ressourcenverbrauch minimieren. Ein Datawarehouse funktioniert meist als Decision Support System (DSS). Das wichtigste sind hier die Reports, welche die Entscheidungsträger benötigen. Die Daten wurden extra für die Reports speziell aufbereitet. Bei einem Datawarehouse geht man deswegen davon aus, dass relativ wenig Benutzer gleichzeitig das Datawarehouse verwenden, dass relativ wenig SQL Statements ausgeführt werden, und dass sehr viele Daten von den Disks gelesen werden. Das bedeutet, dass es relativ wenig Server Processes gibt, dass das Library Cache nicht stark genutzt wird, und dass der Buffer Cache so oder so zu klein ist. *Obwohl also Datawarehouse Applikationen von gut geschriebenen SQL Statements auch profitieren können, liegt der Flaschenhals bei den Input/Output Operationen.*

4.2.1 Ressourcenverbrauch minimieren

Vorgehen:

1. Finden der problematischen SQL Statements
2. Tuning der SQL Statements, so dass die richtigen Indexe verwendet werden
3. Auslagern komplexer Transaktionen in Stored Procedures, um das Netzwerk zu entlasten und den Library Cache besser zu nutzen

SQL Tuning muss zur Ausbildung der Entwickler gehören, damit die Qualität der Software gegeben ist. Im Normalfall wird sich nach der Auslieferung der Software der Kunde früher oder später mit Performance Problemen melden. *Falls diese Probleme nur an einzelnen Stellen auftreten, ist SQL Tuning angesagt.*

4.2.2 Finden der problematischen SQL Statements

Im Normalfall können *Kunden* recht gut beschreiben, unter welchen Umständen ein Performance Problem auftritt. Als Entwickler bleiben dann oft ein, zwei oder zehn SQL Statements übrig, welche potenzielle Verbesserungskandidaten sind.

Falls die langsamen SQL Statements nicht ermittelt werden können, kann man Oracle Trace Dateien erstellen lassen und diese mittels TKPROF analysieren.

Hierzu sind die folgenden Schritte erforderlich:

1. Der INIT.ORA Parameter USER_DUMP_DEST muss auf das Verzeichnis gesetzt werden, in welches die Trace Files geschrieben werden sollen.
2. Der INIT.ORA Parameter MAX_DUMP_SIZE muss auf ein vernünftiges Mass eingestellt werden. Mit diesem Parameter wird die maximale Grösse der Trace Dateien festgelegt.

3. Der INIT.ORA Parameter SQL_TRACE muss auf TRUE gesetzt werden. Dies verlangsamt die Datenbank um 20-30%. Der SQL_TRACE Parameter bestimmt, ob Trace Files geschrieben werden oder nicht.

4. Der INIT.ORA Parameter TIMED_STATISTICS muss auf TRUE gesetzt werden. Die Statistiken können von verschiedenen Analyse Werkzeugen verwendet werden.

Alternativ hierzu kann man diese Parameter auch *pro Session* einstellen. Die Applikation muss hierfür also schon vorbereitet sein. Nun werden die Parameter mit ALTER SESSION gesetzt. Innerhalb von SQL*Plus selber ist dies natürlich nicht sehr interessant, da man dieselbe Information auch mit SET AUTOTRACE ON erhält.

Beispiel für die aktuelle SQL*Plus Session:

```
SQL> connect system/manager@perf
Connected.
SQL> alter system set user_dump_dest='/tmp';

System altered.

SQL> connect scott/tiger@perf
Connected.
SQL> alter session set sql_trace=true;

Session altered.

SQL> alter session set timed_statistics=true;

Session altered.

SQL> select 'Trace Files von Alex' from dual;

'TRACEFILESVONALEX'
--------------------
Trace Files von Alex
```

Das Verzeichnis, welches für USER_DUMP_DEST angegeben wurde, muss für den Oracle Benutzer auf dem Datenbank Server zugänglich sein. Alle Trace Files werden von diesem Oracle Benutzer angelegt und demzufolge *für die meisten anderen Benutzer unzugänglich* sein. Der Oracle Benutzer muss dann auf dem Datenbank Server die Trace Files an die entsprechenden Benutzer weiterleiten (z. B. mail).

Da in das USER_DUMP_DEST Verzeichnis alle Trace Dateien geschrieben werden, und der Name der Datei von Oracle automatisch vergeben wird, sollte man "seine" Dateien *durch ein spezielles SQL Statement identifizieren* können. Im obigen Fall könnte man schlussendlich alle Dateien, welche den String "Trace Files von Alex" enthalten, untersuchen.

Wenn die Trace Files beim Benutzer angekommen sind, kann man diese mittels TKPROF analysieren. Dies produziert eine weitere Datei, in der die Daten aus der Trace Datei neu *formatiert und zusammengefasst* werden.

4.2.3 Tuning der SQL Statements

Die an der Problemstelle ausgeführten *SQL Statements müssen analysiert und gegebenenfalls umgeschrieben werden.* Für SQL Tuning ist kein laufendes System nötig. SQL Tuning kann auf einer Entwicklungsumgebung im Haus stattfinden, solange die Datenvolumen in etwa den Gegebenheiten entsprechen.

Wichtigstes Werkzeug hierfür ist der Execution Plan, den Oracle verwendet, um ein Statement auszuführen.

Mögliche Ineffizienzen:

- Da viele Statements ausgeführt werden, entsteht Netzwerkverkehr und die Datenbank muss wiederholt das Statement Analysieren, aus der Shared SQL Area holen (oder Parsen und den Execution Plan erstellen und diese Information in die Shared SQL Area ablegen). Derartige Statements können kombiniert werden. Vorsicht: Dies reduziert natürlich die Übersichtlichkeit des Codes. Beispiel: SELECT A.EMP_NAME, B.EMP_NAME FROM EMP A, EMP B WHERE A.EMP_NO = 453 AND B.EMP_NO = 263.

- *Full Table Scans werden verwendet, da passende Indexe fehlen.* Vorgehen: Indexe anlegen. Dies ist die wichtigste Regel für grosse Tabellen (bei sehr kleinen Tabellen kann ein Full Table Scan effizienter sein).

- Vorhandene Indexe werden nicht gebraucht. *Falls einzelne Spalten des Indexes in Funktionen verwendet werden, kann der Index nicht mehr verwendet werden.* Beispiel:

  ```
  select * from texte where text like '%wasser%';
  ```

- Typischer Fehler beim Vergleichen von Zeitangaben in Tagen:

  ```
  TRUNC( TRANS_DATE ) = TRUNC( SYSDATE )
  ```

  Dies sollte umgeschrieben werden:

  ```
  TRANS_DATE BETWEEN TRUNC( SYSDATE )
                 AND TRUNC( SYSDATE ) + 0.99999
  ```

  Vorsicht mit dieser Konstante: Falls mehr als fünf Stellen verwendet werden, wird automatisch auf 1 hoch gerundet.

- Typischer Fehler beim Suchen nach String-Anfängen:

  ```
  SUBSTR( ACCOUNT_NAME, 1, 7 ) = 'CAPITAL'
  ```

  Dies sollte umgeschrieben werden:

  ```
  ACCOUNT_NAME LIKE 'CAPITAL%'
  ```

- Typischer Fehler bei der Verwendung von NOT:

```
WHERE DEPT_CODE NOT = 0
```

Dies sollte wenn möglich umgeschrieben werden:

```
WHERE DEPT_CODE > 0
```

Problem: Teilweise kann Oracle diesen Fehler vermeiden indem beispielsweise `NOT >` in `<=` umgewandelt wird. Allerdings gibt es auch negative Überraschungen, indem `!= 0` in `NOT = 0` umgewandelt wird.

- Typischer Fehler bei impliziten Casts: Falls EMP_TYPE ein String ist, so wandelt Oracle bei Vergleichen den Text automatisch um.

```
vorher:   WHERE EMP_TYPE = 123
nachher:  WHERE TO_NUMBER( EMP_TYPE ) = 123
```

Dies sollte umgeschrieben werden:

```
neu:      WHERE EMP_TYPE = '123'
```

Obiger Cast wird von Oracle gemacht, weil Oracle Vergleiche mit numerischen Datentypen bevorzugt.

- Manchmal gibt es mehrere Indexe, welche Oracle verwenden könnte, und Oracle verwendet einen ungünstigeren Index. Diesen kann man dann selektiv ausschalten:

```
AND DEPT_CODE + 0 = 10
```

- Vorhandene Indexe werden in der falschen Reihenfolge gebraucht. Falls der Rule Based Optimizer verwendet wird, kann dieser nicht entscheiden, welcher Index "besser" ist, wenn alle Spalten für zwei Indexe bestimmt sind. Beim Rule Based Optimizer verwendet Oracle dann die Indexe auf die Tabellen zuerst, welche in der FROM Klausel am Ende stehen. Als Daumenregel gilt: *Die Tabellen in der FROM Klausel werden sortiert, so dass die Tabellen am Ende stehen, über die man am meisten weiss.* Falls man ein ER Diagramm vor Augen hat, sind das die Tabellen, "von denen man ausgeht".

- Es kommt zu einem Tree Walk (CONNECT BY). Vorgehen: Indexierung der Spalten in START WITH und CONNECT BY Klauseln.

- GROUP BY falsch angewendet. Vorgehen: Falls möglich GROUP BY vermeiden oder Daten redundant ablegen. Die Pflege der redundanten Daten ist aufwendig und fehleranfällig, also teuer.

- HAVING falsch angewendet. Vorgehen: Falls Datensätze aufgrund von HAVING ausgeschlossen werden, diese mit WHERE ausschliessen.

- Unnötige Verwendung von Views. Vorgehen: Falls möglich die benötigten Daten direkt von den zugrundeliegenden Tabellen holen.

- Unnötige Sortierung wegen DISTINCT. Vorgehen: DISTINCT durch Erweiterung der WHERE Bedingungen vermeiden.
- Unnötige Sortierung wegen UNION. Vorgehen: UNION ALL statt UNION verwenden, falls die Duplikate unproblematisch sind.
- Unnötige Sortierung wegen IN (SELECT ...). Vorgehen: JOIN statt IN (SELECT ...) verwenden, falls das Subselect grosse Datenmengen liefert.

Mögliche Prüfungsfragen

Die Fragen beziehen sich auf folgende Tabellen. Die Systemzeit wird von der Funktion SYSDATE geliefert (auf die Sekunde genau). Der aktuelle Login wird von der Funktion USER geliefert.

```
SQL> desc fh_projekt
 Name                                      Null?    Type
 ----------------------------------------- -------- --------------
 PROJEKT_NR                                NOT NULL NUMBER
 KENNWORT                                           VARCHAR2(20)
 STATUS                                             NUMBER

SQL> desc fh_buchung
 Name                                      Null?    Type
 ----------------------------------------- -------- --------------
 BUCHUNG_NR                                NOT NULL NUMBER
 PROJEKT_NR                                         NUMBER
 USER_NR                                            NUMBER
 STUNDEN                                            NUMBER
 DATUM                                              DATE
 REG_USER                                           VARCHAR2(12)
 REG_DATE                                           DATE
 UPD_USER                                           VARCHAR2(12)
 UPD_DATE                                           DATE

SQL> desc fh_user
 Name                                      Null?    Type
 ----------------------------------------- -------- --------------
 USER_NR                                   NOT NULL NUMBER
 VORNAME                                            VARCHAR2(30)
 NACHNAME                                           VARCHAR2(30)
 TEL_P                                              VARCHAR2(20)
 TEL_G                                              VARCHAR2(20)
 ADR_1                                              VARCHAR2(60)
 ADR_2                                              VARCHAR2(60)
 ADR_3                                              VARCHAR2(60)
 PLZ                                                VARCHAR2(10)
 ORT                                                VARCHAR2(30)
 LAND_NR                                            NUMBER
 LOGIN                                              VARCHAR2(12)
```

- Schreiben Sie ein Select Statement, welches für den heutigen Tag alle Buchungen des aktuellen Benutzers liefert. Jede Buchung soll mit PROJEKT_NR, KENNWORT, STUNDEN, NACHNAHME, VORNAME aufgeführt werden.
- Nehmen wir an, dass beim Anlegen der Tabellen nur Primary Key Constraints angelegt wurden, dh. es existieren nur Indexe auf Primary Keys. Erklären sie, warum ihr Statement keine gute Performance haben wird.
- Welche Indexe würden sie anlegen, um die Performance zu steigern?
- Die Tabelle FH_USER wird für eine kleine Telefonbuch Applikation verwendet. In dieser Applikation gibt es einen Suchen-Dialog, in der man Nachname und/oder Vorname eingeben kann. Diese beiden Angaben sind in den Bind-Variablen :dlgPersonenSuche.dfsNachname und :dlgPersonenSuche.dfsVorname zugänglich. Schreiben sie ein Select Statement, welches Nachname, Vorname, Tel. G und Tel. P für alle gefundenen Benutzer liefert.
- Gehen sie davon aus, dass etwa 20 Mitarbeiter der Firma in der Tabelle FH_USER erfasst wurden. Welchen Index würden sie anlegen, um die Abfrage zu beschleunigen? Diskutieren sie Vor- und Nachteile.
- Gehen sie davon aus, dass etwa 20000 Mitarbeiter der Firma in der Tabelle FH_USER erfasst wurden. Welchen Index würden sie anlegen, um die Abfrage zu beschleunigen? Diskutieren sie Vor- und Nachteile.
- Nehmen sie an, dass die FH_USER Tabelle nicht mehr ganz mit der Applikation übereinstimmt. In der Applikation sind PLZ immer ganze Zahlen zwischen 1000 und 9999. Es gibt einen Mitarbeiter-Suchen Dialog, in dem man eine Zahl eingeben kann. Diese Angabe ist in der Bind-Variable :dlgPersonenAdrSuche.dfnPLZ als Integer zugänglich. Schreiben sie ein Select Statement, welches Nachname und Vorname aller gefundenen Personen liefert.
- Gehen sie davon aus, dass etwa 20000 Mitarbeiter der Firma in der Tabelle FH_USER erfasst wurden. Welchen Index würde sie anlegen, um diese Abfrage zu beschleunigen? Möglicherweise müssen sie das Select Statement abändern, damit der Index verwendet wird. Begründen sie dies.
- In gewissen Fällen kommt es auf die Reihenfolge der Tabellen in der FROM Klausel an. Erklären sie.
- Schreiben sie ein Select Statement, welches für jedes Projekt die in diesem Monat gebuchte Anzahl Stunden liefert. Das Select Statement sollte Projekt-Nr, Kennwort, und Summe aller Stunden zurück liefern. Verwenden sie hierfür die Funktionen LAST_DAY(d), welche den letzten Tag desjenigen Monats liefert, der das Datum d enthält, sowie ADD_MONTHS(d, n), welches zum Datum d weitere n Monate hinzu addiert. n kann auch negativ sein.
- Angenommen es existieren nur Indexe auf Primary Keys. Welche Indexe würden sie anlegen, um die Performance zu steigern? Vielleicht müssen

sie nun die WHERE Bedingung in ihrem Select Statement umschreiben. Begründen sie dies.

- Kommt es im obigen Fall darauf an, welche Reihenfolge die Tabellen in der FROM Klausel haben? Begründen sie.

- Gehen sie davon aus, dass gewisse Kleinstprojekte auf Diplomarbeiten von Studenten basieren. Die Applikation ist hierfür eigentlich nicht ausgelegt, so dass die Anwender angefangen haben, allen Projekte, welche mit Diplomarbeiten zu tun haben, ein Kennwort zu geben, welches mit dem String "DIPLOM" anfängt. Schreiben sie ein Select Statement, welches alle Stunden zusammenzählt, welche im letzten Monat auf Projekte mit Diplomarbeiten gebucht wurden.

- Kommt es im obigen Fall darauf an, welche Reihenfolge die Tabellen in der FROM Klausel haben? Begründen sie.

- Schreiben sie für obigen Fall ein Select Statement, welches alle Stunden zusammenzählt, welche auf Projekte mit Diplomarbeiten gebucht wurden. Im Gegensatz zu vorher ist das Datum nun nicht relevant.

- Kommt es im obigen Fall darauf an, welche Reihenfolge die Tabellen in der FROM Klausel haben? Begründen sie.

Analyse von bestehenden Statements

Analysieren sie die Statements in der Datei example.sql. Beantworten sie hierzu folgende Fragen:

- Welche Indexe würden sie anlegen? Geben sie im Zweifelsfall klar an, ob sie einen Index auf zwei Spalten (die Reihenfolge ist relevant), oder zwei Index auf je eine Spalte anlegen wollen.

- Falls ein Index nur sinnvoll ist, falls die Daten gewisse Eigenschaften haben, geben sie den Index und die nötigen Annahmen an. Formulieren sie die Annahmen so, dass klar wird, wie sie diese Annahmen überprüfen können.

  Beispiel: In der WHERE Bedingung steht "STATUS_ID = 501". Sie vermuten, dass die Spalte STATUS_ID eine geringe Selektivität hat. Ihre Antwort: "Es soll ein Bitmap Index für die Spalte STATUS_ID angelegt werden. Dieser ist nur bei geringer Selektivität interessant. Dies wäre dann der Fall, wenn jeder Wert der STATUS_ID Spalte im Schnitt mehr als hundert Mal vorkommt."

Kapitel 5

Speicher

Die System Global Area (SGA) der Datenbank besteht aus drei Blöcken, welche getuned werden können:

- Shared Pool
  - Library Cache
  - Dictionary Cache
  - Multi Threaded Server Info
- Buffer Cache
- Redo Log Buffer

Zudem gilt es die Grösse des Speichers einzustellen, der für das Sortieren pro Benutzer reserviert wird.

- Sort Area

Das Vorgehen ist wie folgt (solange freier Speicher vorhanden ist):

1. Shared Pool Grösse abschätzen, SHARED_POOL_SIZE eventuell hochsetzen
2. Library Cache Performance überprüfen (ist ein Teil des Shared Pool), SHARED_POOL_SIZE eventuell hochsetzen
3. Data Dictionary Cache Performance (auch Teil des Shared Pool) überprüfen, SHARED_POOL_SIZE eventuell hochsetzen
4. Bei Multi Threaded Server Konfigurationen SHARED_POOL_SIZE eventuell hochsetzen
5. Buffer Cache Performance überprüfen, DB_BLOCK_BUFFERS eventuell hochsetzen
6. Redo Log Buffer Performance überprüfen, LOG_BUFFER eventuell hochsetzen
7. Sort Performance überprüfen, SORT_AREA_SIZE eventuell hochsetzen

5.1 System Global Area (SGA)

Generell gilt, dass *die SGA einerseits so gross* wie möglich sein soll, und dass sich die SGA *andererseits immer im Hauptspeicher* des Servers befinden soll. Neben der Grösse der SGA muss man hierfür natürlich auch die Speicherbelegung der restlichen Prozesse auf dem Server berücksichtigen.

Die Speicherbelegung muss man mit den Tools des Betriebsystems untersuchen. Dies kann je nachdem recht schwer sein, weil ein Betriebsystem wie Linux beispielsweise den freien Speicher für ein Festplatten Cache verwendet. Generell sollten *etwa 5% des Speichers frei* sein. Sind mehr als 5% des Speichers frei, kann man die SGA vergrössern und damit die Performance der Datenbank steigern. Sind weniger als 5% des Speichers frei, besteht die Gefahr, dass Teile des SGA vom Betriebsystem auf die Festplatte ausgelagert werden. Dies führt zu deutlich schlechterer Performance.

Mit dem folgenden Statement wird *die SGA Grösse* angezeigt:

```
SQL> connect system/manager@perf
Connected.
SQL> show sga

Total System Global Area   23666064 bytes
Fixed Size                    64912 bytes
Variable Size               6651904 bytes
Database Buffers           16777216 bytes
Redo Buffers                 172032 bytes
```

Generell gilt, dass man, bevor man die Grössen der einzelnen Teile ändert, *die Performance von Library Cache, Buffer Cache und Redo Log Buffers überprüfen* sollte.

5.1.1 Die richtige Shared Pool Grösse abschätzen

Eine Möglichkeit, die *richtige* Grösse abzuschätzen:

1. SHARED_POOL_SIZE auf einen *sehr* grossen Wert setzen
2. Die Datenbank eine Weile lang benutzen
3. Die gewünschte Grösse des Shared Pools berechnen, indem man folgende Werte addiert und aufrundet:

   - *Packages, Views*, etc.:

     ```
     SQL> select sum(sharable_mem) from v$db_object_cache;

     SUM(SHARABLE_MEM)
     -----------------
               1239074
     ```

   - *SQL Statements*:

```
SQL> select sum(sharable_mem) from v$sqlarea
     where executions > 5;

SUM(SHARABLE_MEM)
-----------------
           236724
```

- *Pro Benutzer*:

```
SQL> select sum(250 * users_opening) from v$sqlarea;

SUM(250*USERS_OPENING)
----------------------
                   250
```

- *Pro Benutzer in der Multi Threaded Server Konfiguration* (an dieser Stelle muss ein Wert zwischen dem aktuellen und dem maximalen Wert gewählt werden):

```
SQL> select sum(value) from v$sesstat, v$statname
     where name = 'session uga memory'
     and v$sesstat.statistic# = v$statname.statistic#;

SUM(VALUE)
----------
    229932

SQL> select sum(value) from v$sesstat, v$statname
     where name = 'session uga memory max'
     and v$sesstat.statistic# = v$statname.statistic#;

SUM(VALUE)
----------
    268988
```

5.2 Library Cache Performance überprüfen

Wird Statement oder Package im Library Cache gefunden, kann der Parsing Schritt übersprungen werden. Wenn ein Statement oder Code aus einer PL/SQL Package benötigt wird, wird zuerst geprüft, ob sich Statement oder Package schon im Library Cache befinden. Um dies zu prüfen wird ein einfacher String-Vergleich gemacht.

Um die Performance des Library Caches zu überprüfen, verwendet man die V$LIBRARYCACHE View.

Mit folgendem Statement wird das Verhältnis von PINS zu RELOADS angezeigt. Die Performance ist schlecht, falls es zu mehr als 1% Reloads kommt. Ein PIN ist ein Cache-Hit, ein RELOAD ein Cache-Miss. *Ein RELOAD bedeutet, dass ein Statement ein zweites Mal geparst werden musste oder dass es neu geladen werden musste, weil es aus dem Library Cache entfernt wurde.* Auf jeden Fall bedeuted es, dass man die Library Cache Grösse hinauf setzen sollte.

```
SQL> select sum(pins) pins,
            sum(reloads) reloads,
            sum(reloads)/(sum(pins)+sum(reloads)) ratio
     from v$librarycache;

      PINS    RELOADS      RATIO
---------- ---------- ----------
      2149         10 .004631774
```

Grundsätzlich gilt es, möglichst alle RELOADS zu vermeiden. Um die RELOADS zu reduzieren, *erhöht man den ORA.INIT Parameter SHARED_POOL_SIZE.*

5.2.1 Den Cache richtig verwenden

Der Oracle verwendet einen String-Vergleich, um SQL Statements im Library Cache zu finden. Gross- und Kleinschreibung, Tabulatoren, Space, Zeilenumbrüche etc. sind für den Vergleich relevant! *Dieser String-Vergleich funktioniert allerdings auch dann, wenn Statements Bind Variables enthalten.* Im folgenden Beispiel kann das geparste Statement aus dem Library Cache verwendet werden, solange nur die Person oder die Sprache ändern. Ändert sich allerdings einer der Adressentypen oder die Art der Personen-Firmen-Beziehung, wird sich das Statement von demjenigen im Library Cache unterscheiden.

```
SELECT  P.NAME, P.VORNAME, C.NAME, CA.CITY, L.NAME
FROM    LAND L,
        ADDRESS A,
        COMPANY_ADDRESS CA,
        COMPANY C,
        PERSON_COMPANY PC,
        PERSON P
WHERE   P.PERSON_NR = :dlgExplorer.nOutlineKey
AND     PC.PERSON_NR = P.PERSON_NR
AND     C.COMPANY_NR = PC.COMPANY_NR
AND     PC.TYPE = 1
AND     CA.ADDRESS_NR = C.ADDRESS_NR
AND     CA.TYPE = 2435
AND     A.ADDRESS_NR = CA.ADDRESS_NR
AND     A.TYPE_NR = 645
AND     L.LAND_NR = A.LAND_NR
AND     L.LANGUAGE_ID = :User.nLanguage
```

Hieraus ergibt sich die Anforderung an alle Entwickler, einen *einheitlichen SQL Stil* zu pflegen, damit möglichst viele SQL Statements im Library Cache wiederverwendet werden können.

5.2.2 Die Alternative: Stored Procedures

Werden komplexe Transaktionen als Stored Procedure auf der Datenbank abgelegt, so *spart dies zudem Netzwerkverkehr* zwischen Client und Server, da nur der Aufruf der Stored Procedure und nicht alle SQL Statements übermittelt

werden müssen. Stored Procedures können nicht verwendet werden, wenn Select Statements mehr als eine Zeile liefern oder wenn vom Benutzer interaktiv Entscheidungen verlangt werden.

Beispiel 1: Wird einer Kundenauftragsposition ein Artikel und eine Spezifikation zugewiesen, *sollen beim Speichern die Schilderdaten auf Basis der Spezifikationsdaten und der zum Artikel angelegten Schildervorlagen generiert werden.* Dies sind ideale Voraussetzungen, um den gesamten Prozess in eine PL/SQL Package auszulagern, da es fast keine Kommunikation zwischen Client und Server gibt.

Beispiel 2: Eine Person kann bei mehreren Firmen arbeiten. In einem normalen SELECT Statement *soll in Klammern die Liste der Arbeitgeber zu jeder Person angezeigt werden.* Das Resultat kann man sich nicht von einer PL/SQL Funktion liefern lassen; eine Spalte kann man sich allerdings sehr wohl von einer PL/SQL Funktion liefern lassen. Dies zeigt auch, wo das Problem liegt: will man nun in einer weiteren Spalte die jeweiligen Standorte der Arbeitgeber auflisten, kann man diese Spalte wohl auch von einer PL/SQL Funktion liefern lassen, doch wird die Funktion *Liste der Arbeitgeber Standorte* völlig unabhängig von der Funktion *Liste der Arbeitgeber* sein.

Beispiel 3: Bevor der Status eines Fertigungsauftrages geändert werden kann, soll eine komplexe Prüfung der entsprechenden Bedingungen durchgeführt werden. Falls diese Prüfung ergibt, dass der Status nicht geändert werden darf, *soll der Benutzer mit einer entsprechenden Erklärung informiert werden.* Dies lässt sich nur sehr mühsam auf PL/SQL übertragen, da die Fehlermeldungen selber auf der Client-Seite erzeugt werden müssen. Dies bedeutet, dass die möglichen Ereignisse auf der Server-Seite im PL/SQL festgelegt werden, dass die entsprechenden Reaktionen aber auf der Client-Seite in der entsprechenden Programmiersprache ausprogrammiert werden müssen. Das ist fehleranfällig und unübersichtlich.

5.3 Data Dictionary Cache Performance überprüfen

Oracle verwendet den eigenen Data Dictionary Cache ständig. Um zu überprüfen, ob der Shared Pool gross genug für den Data Dictionary Cache ist, überprüft man hier (wie beim Library Cache) die Hits.

Um die Performance des Data Dictionary Caches zu überprüfen, verwenden man die V$ROWCACHE View.

```
SQL> select (1-(sum(getmisses)/(sum(gets)+sum(getmisses)))) ratio
     from v$rowcache;

     RATIO
----------
.857018309
```

Die Trefferrate sollte über 90% liegen. Nach dem Hochfahren der Datenbank ist der Data Dictionary Cache allerdings noch leer. Er wird erst nach und nach gefüllt. Deswegen wird nach dem Hochfahren der Datenbank die Trefferrate besonders tief sein. Hier, wie auch bei allen anderen Performance relevanten

Abfragen, ist es wichtig mit einem System zu arbeiten, dass schon seit einiger Zeit läuft.

Einige der Einträge in V$ROWCACHE werden besonders oft gebraucht. Entsprechend wichtig ist dort die Trefferrate: *dc_user_grants sollte bei 0.95 und dc_users bei 0.70 liegen.*

```
SQL> select parameter,
            (1-getmisses/(gets+getmisses)) ratio
     from v$rowcache
     where parameter in ('dc_user_grants', 'dc_users')
     and gets+getmisses != 0;

PARAMETER                          RATIO
-------------------------------- ----------
dc_users                        .745762712
dc_user_grants                        .675
```

Wenn das System schon eine Weile läuft, und die Data Dictionary Trefferrate trotzdem zu tief liegt, sollte der ORA.INIT Parameter SHARED_POOL_SIZE erhöht werden. Dies stellt dem Library Cache und dem Data Dictionary Cache mehr Platz zur Verfügung.

5.4 Multi Threaded Server (MTS) Information im Shared Pool

Wird die Datenbank in der Multi Threaded Server Konfiguration betrieben, so arbeiten die Server Prozesse die SQL Statements nicht mehr pro User ab. Gewisse User Informationen müssen nun allen Servern zugänglich sein und können nicht mehr in der User Global Area (UGA) abgelegt werden. Diese Informationen werden dann im Shared Pool abgelegt.

5.5 Buffer Cache Performance überprüfen

Grundsätzlich sollte der Buffer Cache erst getuned werden, wenn der Shared Pool getuned worden ist.

Wird ein DB Block im Buffer Cache gefunden, kann der entsprechende Plattenzugriff vermieden werden. Deswegen ist es von Vorteil, wenn der Buffer Cache so gross wie möglich ist. Da die Datenbank aber kaum vollständig in den Speicher des Servers geladen werden kann, werden die anderen Parameter getuned, bevor der Buffer Cache vergrössert wird.

Um die Trefferrate des Buffer Caches zu überprüfen, verwendet man die V$SYSSTAT View.

Mit folgendem Statement wird die Information angezeigt, welche man benötigt, um die *Trefferrate* zu berechnen:

```
SQL> select name, value
     from v$sysstat
     where name in ('db block gets',
                    'consistent gets',
```

```
                    'physical reads');

NAME                                                    VALUE
------------------------------------------------------ ----------
db block gets                                             627
consistent gets                                         12047
physical reads                                           1597
```

Die Trefferrate *berechnet sich wie folgt*:

```
Trefferrate = (logical reads - physical reads) / (logical reads)
wobei logical reads = consistent gets + db block gets
```

In unserem Fall also:

```
logical reads = 627 + 12047 = 12674
Trefferrate = (12674 - 1597) / 12674 = 0.874
```

Die Performance ist gut, falls die Trefferrate über 0.95 liegt.

Eine weitere Möglichkeit die Performance des Buffer Caches zu überprüfen, findet man in der V$SYSTEM_EVENT View: Wenn *buffer busy waits* hoch ist, kann dies bedeuten, dass der Buffer Cache der Datenbank zu knapp bemessen ist.

Um die Performance des Buffer Caches zu verbessern, *erhöht man den ORA.-INIT Parameter DB_BLOCK_BUFFERS*.

Falls man überprüfen will, ob das Betriebssystem die Instanz bremst, kann man V$SYSTEM_EVENT überprüfen. Interessant ist, wenn *buffer busy waits* hoch ist, kann dies bedeuten, dass das buffer cache der Datenbank zu knapp bemessen ist. Dieselben Informationen findet man pro Session auch in V$SESSION_EVENT und V$SESSION_WAIT.

5.6 Redo Log Buffer

Der default Wert für die Redo Log Buffer Grösse ist 4x die ORACLE Blockgrösse. Diese Blockgrösse ist Plattform abhängig. Für ein stark belastetes System ist dieser default Wert oft viel zu klein. Den grössten Einfluss haben Redo Logs allerdings, wenn sie in die Redo Logs gespeichert werden, denn dann handelt es sich um IO Operationen.

Um die Performance der Redo Log Buffers zu überprüfen, verwendet man die V$SYSSTAT View. Es ist allerdings nicht ganz klar, welcher Parameter nun tatsächlich am aussagekräftigsten ist:

- Im Handbuch *Oracle 8 Tuning* heisst es, dass das Verhältnis *redo log space requests : redo entries kleiner als 1:5000* sein sollte. Ist es grösser, soll die Redo Log Buffer Grösse hochgesetzt werden.
- Im Buch *Oracle Performance Tuning* von Corrigan und Gurry heisst es, dass *redo log space wait time null* sein sollte.
- In den *Oracle Performance Tuning* Schulungsunterlagen von Oracle heisst es, dass *redo buffer allocation retrieval null* sein sollte.

Mit folgendem Statement werden alle entsprechenden Parameter angezeigt:

```
SQL> select name, value
     from v$sysstat
     where name in ('redo buffer allocation retries',
                    'redo log space wait time',
                    'redo log space requests',
                    'redo entries');

NAME                                                     VALUE
-------------------------------------------------------- ----------
redo entries                                                 42
redo buffer allocation retries                                0
redo log space requests                                       0
redo log space wait time                                      0
```

Egal welche Parameter man verwendet, um das Problem zu diagnostizieren, die Lösung ist immer gleich. Um den Redo Log Buffer zu vergrössern, *erhöht man den ORA.INIT Parameter LOG_BUFFER.*

5.7 Sort Area

Die Sort Area ist ein Speicherbereich, der *pro Benutzer für Sortiervorgänge* reserviert wird.

Um die Nutzung der Sort Area zu überprüfen, verwendet man die V$SYSSTAT View.

Mit folgendem Statement werden die Anzahl Sortiervorgänge aufgeführt, die *mit bzw. ohne Plattenzugriffe* durchgeführt wurden. Da die Performance unter Plattenzugriffen leidet, gilt es, diese zu reduzieren.

```
SQL> connect system/manager@perf
Connected.
SQL> select name, value
     from v$sysstat
     where name like 'sort%';

NAME                                                     VALUE
-------------------------------------------------------- ----------
sorts (memory)                                              146
sorts (disk)                                                  0
sorts (rows)                                               1129
```

Um die Sortiervorgänge auf der Platte zu reduzieren, kann man entweder die SQL Statements umschreiben, so dass weniger Daten sortiert werden, oder man *erhöht den ORA.INIT Parameter SORT_AREA_SIZE.*

Folgende SQL Befehle bewirken einen Sortiervorgang:

- CREATE INDEX
- DISTINCT

- GROUP BY
- ORDER BY
- UNION, INTERSECT, MINUS
- Join von nicht indexierten Tabellen
- Bestimmte Subselects

Sortiervorgänge lassen sich besonders einfach vermeiden, wenn diese aufgrund von *fehlenden Indexen* passieren. In diesem Fall legt man einfach den fehlende Index an. Schwieriger wird es, wenn man im Execution Plan erkennt, dass die Daten *mehrmals sortiert* werden. In vielen Fällen kann man durch Umschreiben des SQL Statements ein paar dieser Sortiervorgänge einsparen.

Lassen sich Sortiervorgänge auf der Platte nicht vermeiden, dann ist entscheidend, *wo* die temporären Segmente angelegt werden, *und wie gross* INITIAL und NEXT Extents für *die temporären Segmente sind*. Anscheinend ist 1 + n * SORT_AREA_SIZE Blocks eine gute Wahl für die Grösse der Extents. Derartige Details sind aber nur selten von Bedeutung; nur selten lassen sich bedeutenende Performance Steigerungen in diesem Bereich erzielen.

Beispiel:

```
SQL> connect system/manager@perf
Connected.
SQL> select value/1024
     from v$parameter
     where name like 'db_block_size';

VALUE/1024
----------
         2

SQL> select value / 1024
     from v$parameter
     where name like 'sort_area_size';

VALUE/1024
----------
        64
```

Wenn die Sort Area Size also vergrössert wird, empfiehlt sich eine Grösse von 66K, 130K, oder gar 194K.

Um den Speicherverbrauch für ungenutzte Sort Areas bei vielen gleichzeitigen Benutzern zu minimieren, kann man den INIT.ORA Parameter SORT_AREA_RETAINED_SIZE einstellen. Empfohlen wird bei Speicherknappheit die Hälfte von SORT_AREA_SIZE.

Bei eine Grösse von 128K für die SORT_AREA_SIZE, könnte man also SORT_AREA_RETAINED_SIZE auf 64K einstellen.

Kapitel 6

I/O

Grundsätzlich gelten die folgenden Grundsätze:

- Die Daten sollen mit möglichst wenig Plattenzugriffen gelesen und geschrieben werden. Hierfür werden Indexe verwendet.
- Die Plattenzugriffe sollten möglich gut auf die verschiedenen Platten verteilt werden, damit die Zugriffe so weit es geht parallel ablaufen können. Dies ist für die parallele Verarbeitung von SQL Statements essenziell. Dies gilt nicht nur für die Daten (Tabellen und Indexe) sondern auch für Tablespaces, Rollback Segmenten und Redo Log Files.

Der Rest des Kapitels beschäftigt sich allerdings nicht mit der Optimierung der Festplattennutzung sondern mit Technologien, welche den Zugang über das Netzwerk verbessern und vorhandene Rechenleistung besser nutzen.

Multi Threaded Server (MTS) Statt für jeden Benutzer einen Dedicated Server Process zu starten, werden mehrere Benutzer von einem Shared Server Process bedient. Der Datenbank Server muss als Multi Threaded Server konfiguriert werden. Die Multi Threaded Server Konfiguration kann *mehr Benutzer gleichzeitig* bedienen, da auf dem Server weniger Prozesse gestartet werden müssen und somit weniger Speicher verbraucht wird.

Parallel Execution SQL Statements können von mehreren Prozessen gleichzeitig ausgeführt. Der Server Prozess, welcher ein Statement abarbeitet, kann die Arbeit dann aufteilen und weitere Prozesse starten, die sich der Arbeit annehmen. Dies ist nur im Rahmen von Oracle8 Enterprise Edition verfügbar. Parallel Execution bietet sich für *Symmetric Multiprocessing (SMP) Hardware* an, dh. bei Mehrprozessorsystemen mit *einem* Speicher. Vorteil: Mehrere CPUs können an einem Statement arbeiten. Nachteil: Mehr CPUs, Speicher, und Platten sind teuer.

Oracle Parallel Server (OPS) Option Dies ist eine zusätzliche Option, welche für Oracle8 Enterprise Edition installiert werden kann. OPS erlaubt Parallel Execution von SQL Statements mit *Cluster und Massively Parallel Processing (MPP) Hardware*, dh. bei Systemen mit mehr als einem Speicher. Die Prozessoren mit ihrem jeweiligen Speicher werden *Knoten* genannt. Vorteil: Noch mehr CPUs können parallel an einem Statement

arbeiten. Nachteil: Die Verbindungen zwischen den einzelnen Knoten bildet einen Flaschenhals, der nicht transparent ist.

Verteilte Datenbanken Man kann mehrere Datenbanken auch vernetzen. In diesem Fall werden zwar die Daten selber ausgetauscht, es ist aber keine besondere Hardware nötig. Vorteil: Leicht zu verwenden. Nachteil: Wartung wird schnell sehr aufwendig, die Verbindungen zwischen den einzelnen Datenbanken bildet weiterhin einen Flaschenhals, der nicht transparent ist.

Allen drei Varianten ist gemeinsam, dass sie *kompliziert zu installieren und schwierig zu warten* sind.

6.1 Multi Threaded Server

Die Multi Threaded Server Konfiguration ist *für Benutzer und Entwickler transparent*, dh. an Applikationen und SQL Statements muss nichts verändert werden.

Die Multi Threaded Server Konfiguration bringt insbesondere dann etwas, *wenn die Benutzer der Datenbank nicht sehr aktiv sind.* Statt auf dem Server für jeden Benutzer einen Dedicated Server Process zu starten, werden mehrere Benutzer bzw. Netzwerkverbindungen von einem Shared Server Process bedient.

Oracle empfiehlt *einen Shared Server Processes pro zehn Netzwerkverbindungen.* Die Anzahl Shared Server Processes wächst dynamisch, sinkt aber nie unter eine gewisse untere Grenze. Diese Grenze wird mit dem MTS_SERVERS Parameter eingestellt. Diesen Parameter kann man bei laufenden System mit ALTER SYSTEM ändern. Die obere Grenze wird mit dem MTS_MAX_SERVERS Parameter eingestellt. Dieser Parameter kann nur in der INIT.ORA Datei eingestellt werden.

Der Shared Server Process ist nicht mehr dafür verantwortlich, das Resultat von SQL Statements an den Benutzer zurück zu übermitteln. Dies übernimmt ein Dispatcher Process. Oracle empfiehlt *einen Dispatcher Process pro 250 Netzwerkverbindungen.* Die Anzahl Dispatcher Processes ist fix und wird mit dem MTS_DISPATCHERS Parameter eingestellt.

Mit der Multi Threaded Server Konfiguration erlaubt Oracle *Connection Pooling, Connection Multiplexing, und Connection Load Balancing.* Damit werden einerseits inaktive Verbindungen zu Clients temporär unterbrochen, und andererseits werden Verbindungen über einen Connection Manager geführt, welcher mehrere Verbindungen annehmen kann und nur jeweils eine an den Datenbank Server weiter leitet. Der Listener Process, welcher neue Verbindungen entgegennimmt, betreibt Connection Load Balancing indem er neue Verbindungen an den Dispatcher Process weiterleitet, welcher gerade am wenigsten zu tun hat.

Der leitet die Anfragen an einen Shared Server Process weiter und liefert das Resultat an den jeweiligen Benutzer zurück. Statt also für n Benutzer n Dedicated Server Processes zu verwenden, werden für n Benutzer ca. n / *250* Dispatcher Processes und n / *10* Shared Server Processes verwendet.

Details: Wenn ein Benutzer eine Verbindung startet, wird diese vom Listener an den Dispatcher Process vermittelt, der am wenigsten zu tun hat. Wenn der Benutzer ein SQL Statement ausführt, nimmt der Dispatcher dieses entgegen, und legt es auf die

Request Queue, welche von allen Dispatchern verwendet wird. Sobald ein Shared Server Process frei ist, nimmt er das nächste SQL Statement und führt es aus. Das Resultat wird vom Shared Server Process in die Response Queue des Dispatcher Processes geschrieben, welcher die Anfrage geschickt hat. Der Dispatcher prüft seine Response Queue und schickt das Resultat an den Benutzer zurück, der das SQL Statement geschickt hat.

Mögliche Prüfungsfragen

- Der Lieferant des neuen CRM Systems, welches sie in ihrer Firma einsetzen wollen, empfiehlt ihnen, Oracle in der Multi Threaded Server (MTS) Konfiguration einzusetzen. Als Begründung führt der Lieferant an, dass die Performance somit gesteigert werden kann. Unter welchen Umständen hat der Lieferant recht?

6.2 Parallel Execution

Parallel Execution ist *für Benutzer und Entwickler transparent*, dh. an Applikationen und SQL Statements muss nichts verändert werden.

Parallel Execution eignet sich insbesondere *für Datawarehouse Anwendungen*. Da die einzelnen SQL Statements sehr aufwendig sind, kann die Arbeit auf mehrere Prozessoren aufgeteilt werden. Hierfür wird die Arbeit in Einzelschritte aufgeteilt, so dass die Daten wie in einer UNIX Pipe von einem Prozess zum nächsten fliessen können. So können die nachfolgenden Prozesse mit ihrer Arbeit anfangen, obwohl ihre vorangehenden Prozesse mit der Arbeit noch nicht fertig sind.

Parallel Execution bietet *für den Normalbetrieb eines OLTP Systems keine Vorteile*, da in einem OLTP System die einzelnen SQL Statements sehr einfach sind. Parallel Execution für OLTP Anwendungen eignet sich nur, wenn über Nacht grosse Datenmengen bearbeitet werden müssen.

Parallel Execution bietet nur dann Vorteile, *wenn folgende Bedingungen alle erfüllt sind*:

1. Genügend CPUs, dh. Symmetric Multiprocessing (SMP) Hardware, Cluster Hardware, oder Massively Parallel Processing (MPP) Hardware
2. Genügend I/O Bandbreite
3. Genügend CPU Kapazität (z. B. meist weniger als 30% Auslastung)
4. Genügend Speicher für weitere speicherintensive Prozesse wie Sortieren, Hashen oder I/O Buffers

Unter Oracle8 ist es möglich, *Parallel Execution Tuning automatisch durchführen* zu lassen. Hierfür setzt man den PARALLEL_AUTOMATIC_TUNING Parameter auf TRUE. Alle Parameter, welche Parallel Execution beinflussen, werden dann automatisch verwaltet. Auf das manuelle Tuning von Parallel Execution wird nicht weiter eingegangen.

Um Parallel Execution zu *aktivieren*, gibt man entweder die *PARALLEL Option* beim CREATE TABLE oder ALTER TABLE Statement an, oder man verwendet in SQL Statements den *PARALLEL Hint*.

Das Hauptproblem ist die Synchronisation. Je mehr Synchronisation zwischen einzelnen Arbeitsschritten nötig ist, um so schlechter die Performance. Das Hauptproblem liegt darin, die Arbeit so aufzuteilen, dass zwischen einzelnen Arbeitsschritten so wenig Synchronisation wie möglich statt findet.

Oracle bildet Arbeitsschritte nach folgenden Regeln, in der Hoffnung die Synchronisation reduzieren zu können:

1. Block Bereiche: Dies wird für Select Statements und Subselects verwendet, oft in Kombination mit Partitionierung (nächste Regel).

2. *Partitionierung* von Tabellen oder Indexen: Dies wird für Insert, Update, Delete, Direct-Load, Create Index und Create Table Statements verwendet. Die Partitionierung kann entweder nach Bereichen einzelner Attribute geschehen (ideal für historische Daten), nach Hash Werten (garantiert eine gleichmässige Verteilung der Daten) oder nach einer Kombination der beiden genannten Methoden (Partitionierung nach Attribut Bereich, Subpartitionierung nach Hash Wert).

3. *Striping* der Tablespaces: Dies kann entweder vom Betriebsystem oder von Oracle gemacht werden. Hier bietet sich RAID5 an, da die Daten so zudem redundant gespeichert werden. Da RAID Systeme Daten etwas langsamer schreiben als alleinstehende Disks, können RAID Systeme für gewisse OLTP Anforderungen zu langsam sein. Für Temporäre Tablespaces lohnt sich zwar Striping, aber kaum ein RAID System. Mehr zu Striping auf Seite 65.

Weitere Details hierzu findet man im Oracle Tuning Handbuch. Folgende SQL Statements sollten von Parallel Execution profitieren:

- table scan
- nested loop join
- sort merge join
- hash join
- “not in”
- group by
- select distinct
- union and union all
- aggregation
- PL/SQL functions called from SQL
- order by
- create index
- update
- delete

- insert ... select
- *etc.*

Select Statements, welche Indexe verwenden, profitieren nicht direkt von Parallel Execution.

Mögliche Prüfungsfragen

- Unter welchen Umständen lohnt es sich, Parallel Execution zu aktivieren?

6.3 Oracle Parallel Server

Oracle Parallel Server (OPS) nutzt die Möglichkeiten von Cluster und Massively Parallel Processing (MPP) Hardware. Hiermit können mehrere Server (jeweils mit mehreren CPUs und eigenem Hauptspeicher) zusammen arbeiten. Die einzelnen Servers werden Knoten (nodes) genannt.

OPS bietet sich an, wenn das Verarbeiten der Daten sehr viel CPU und Disk Ressourcen bindet, und *wenn die Kommunikation zwischen den einzelnen Knoten minimiert werden kann.* In diesem Fall können einzelne Aufgaben von verschiedenen Knoten bearbeitet werden. Da die Kommunikation zwischen den einzelnen Servern allerdings problematisch werden kann, profitieren nur wenig Applikationen von einer Parallel Server Option:

- wenn Daten vor allem nur gelesen werden
- wenn vor allem Daten geändert werden, die auch physikalisch getrennt voneinander vorliegen
- wenn dieselben Daten zu unterschiedlichen Zeiten geändert werden

Die Applikationen müssen für OPS angepasst werden. Die Kosten für das Redesign der Datenbank und das Tuning der Parallel Server Umgebung können sehr gross werden. Meistens ist es günstiger, den bestehenden Server aufzurüsten.

6.4 Verteilte Datenbanken

Wenn Wartezeiten wegen dem Netzwerk nicht allzusehr ins Gewicht fallen, gibt es eine einfache Alternative zu OPS Konfiguration: Verteilte Datenbanken. Jede der Datenbanken muss zwar eigenständig gepflegt werden, oft ist diese Aufgabe aber bei weitem nicht so anspruchsvoll und wie die Wartung einer grossen OPS Konfiguration.

Es gibt verschiedene Möglichkeiten, Daten zu verteilen:

- DB-Links: Mit einem DB-Link kann man Tabellen lesen und bearbeiten, welche auf einem anderen Server liegen. DB-Links können mittels einem Gateway auch auf Datenbanken anderer Anbieter zugreifen (z. B. DB2).
- Materialized View: Mit einer Materialized View wird eine lokale Kopie einer Tabelle erstellt, welche in regelmässigen Abständen von Oracle aufgefrischt wird. Eine Materialized View wird auch ein *Snapshot* genannt. Der Datenaustausch Prozess wird *Replikation* genannt.
- Proprietäre Replikation: In gewissen Fällen lohnt es, eine eigene Replikation zu schreiben.

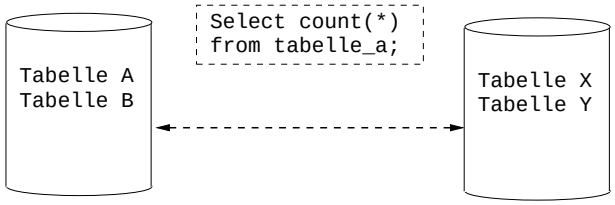

Abbildung 6.1: DB Link

6.5 DB Link

Ein DB Link funktioniert wie *ein SQL Statement Proxy*: Der DB Link nimmt SQL Statements entgegen, meldet sich bei der anderen Datenbank an, führt das SQL Statement dort aus, und liefert das Resultat zurück.

```
SQL> create database link autoperf connect to scott
     identified by tiger using 'perf';

Database link created.

SQL> set autotrace on explain
SQL> select count(*)
     from emp e
     where not exists (
       select 1
       from emp@autoperf re
       where e.ename = re.ename );

  COUNT(*)
----------
         0

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   SORT (AGGREGATE)
   2    1     FILTER
   3    2       TABLE ACCESS (FULL) OF 'EMP'
   4    2       REMOTE*                                AUTOPERF

   4 SERIAL_FROM_REMOTE            SELECT "ENAME" FROM "EMP" "RE"
                                   WHERE :1="ENAME"

SQL> drop database link autoperf;

Database link dropped.
```

Die Daten, welche über einen DB Link von der remote Datenbank geholt werden, sind *immer aktuell*. Je nach Netzwerkverbindung kann die Performance allerdings miserabel sein.

DB Links können entweder den aktuellen Username und dessen Passwort verwenden, um auf der remote Datenbank einzusteigen, oder Username und Passwort können fix sein. Im obigen Beispiel Username und Passwort fix. *Um den Zugriff auf die Tabellen der remote Datenbank einzuschränken, müssen auf dem lokalen System Views angelegt werden, auf die lokale Benutzer dann zugreifen.* Den lokalen Benutzern bzw. den lokalen Rollen kann man dann Grants auf diese Views geben.

Trotz diesem zusätzlichen Wartungsaufwand ist die Verwendung von DB Links *um einiges einfacher als die Verwendung der Oracle Parallel Server Option*, aber auch entsprechend weniger mächtig: Die Arbeit eines SQL Statements kann nicht mehr auf mehrere Rechner verteilt werden. Nur die Daten werden verteilt.

Da die Verwendung des DB Links mittels Synonymen versteckt werden kann, sind DB Links für die Benutzer und Entwickler *transparent*. Einzige Ausnahme: Performance Probleme bei Netzwerk Flaschenhälsen. Im folgenden Beispiel werden nun alle Statements für EMP auf der VERKAUF Datenbank mit dem SCOTT/TIGER account ausgeführt, da EMP ein Synonym ist.

```
SQL> CREATE DATABASE LINK verkauf_link
     CONNECT TO scott IDENTIFIED BY tiger
     USING 'verkauf';

SQL> SELECT *
     FROM emp@verkauf_link;

SQL> CREATE SYNONYM emp
     FOR scott.emp@verkauf_link;

SQL> SELECT *
     FROM emp;
```

In diesem Zusammenhang wird oft der Begriff *Two Phase Commit* verwendet. Der Two Phase Commit garantiert dass eine Transaktion, die mehrere Datenbank betrifft, auf allen beteiligten Datenbanken gleich behandelt wird: ein Commit oder ein Rollback wird immer für *alle* beteiligten Datenbanken ausgeführt. Der Two Phase Commit findet hinter den Kulissen statt und ist für die Benutzer transparent, genauso wie viele anderen Probleme, welche sich in diesem Zusammenhang stellen (Security, Distributed Query).

Mögliche Prüfungsfragen

- Für einen Kunden mit Niederlassungen in Genf, Zürich und Basel haben sie eine Applikation entwickelt, welche eine Datenbank verwendet. An allen drei Niederlassungen befindet sich eine Datenbank. Damit die Daten der anderen Niederlassungen jeweils auch einsehbar sind, empfiehlt ihnen der System Engineer, einen DB-Link zu den remote Systemen anzulegen. Was wären Vor- und Nachteile dieser Lösung? Überlegen sie, unter welchen Umständen ein Snapshot angebracht wäre. Stellen sie eine kurze Liste von Fragen zusammen, welche sie an der nächsten Kernteamsitzung dem Kunden stellen wollen, um diese Frage zu entscheiden.

Abbildung 6.2: Snapshot

- Erklären sie in zwei oder drei Sätzen, warum ein DB-Link eine gute Lösung wäre, um die Daten einer remote Datenbank lokal zugänglich zu machen.
- Diskutieren sie Vor- und Nachteile eines DB-Links.

6.6 Materialized Views / Snapshots

Falls die Daten nicht aktuell sein müssen, kann man eine Materialized Views verwenden. Eine Materialized View ist eine View, welche die Daten der Originaltabelle physikalisch kopiert und diese dann regelmässig nachführt. Eine Materialized View kann read-only oder updatable sein. Ist die Materialized View updatable, werden Änderungen in beide Richtungen ausgetauscht. *Bei Konflikten werden Logeinträge geschrieben, welche in irgendeiner Weise behandelt werden müssen.* Dies kann durch einen Datenbank Administrator (DBA) oder programmatisch geschehen (z. B. in regelmässigen Abständen mittels Datenbank Jobs).

Eine Materialized View kann aus Performance Gründen für lokale Tabellen angelegt werden, falls gewisse Aggregationen oder Joins sehr aufwending sind. In diesem Fall werden Materialized Views oft bei jeder Änderung der original Daten nachgeführt.

Wurde die Materialized View für remote Tabellen angelegt, spricht man auch von einem *Snapshot*. Beim Anlegen der Materialized View muss angegeben werden, mit welchen Abständen oder unter welchen Umständen der Abgleich mit den original Daten erfolgt. Diesen Vorgang nennt man *Replikation*.

Werden die Daten der remote Tabellen nur selten geändert, kann man den Snapshot auch sofort nachführen lassen. Man redet dann von *Synchronous Data Replication* oder von *Real-Time Data Replication*. Dies reduziert mögliche Replikationskonflikte.

Eine Replikation ist für Benutzer *nicht transparent*, da Änderungen nur langsam propagiert werden.

Mögliche Prüfungsfragen

- Die Verkaufsleiter des Kunden wünschen sich für die Auswertungen der Verkaufstätigkeiten die aktuellsten Daten aus allen 25 Niederlassungen. Ihr DBA schlägt vor, dies über Materialized Views zu lösen. So können sie am Hauptsitz die Daten aller 25 Niederlassungen einsehen, da alle mit denselben Daten arbeiten. Für die Spezifikation der Lösung müssen sie nun entscheiden, ob sie Real-Time Data Replication oder Synchronous Data Replication verwenden wollen. Bei der nächsten Sitzung mit dem

Kunden müssen sie nun kurz die Vor- und Nachteile der beiden Varianten aufzeigen und dem Kunden eine Empfehlung abgeben. Was sagen sie?

6.7 Proprietäre Replikation

Alle Änderungen einer Tabelle werden mittels Triggers in Journal Tabellen festgehalten. Aus diesen Journal Tabellen werden dann später dynamisch SQL Scripts generiert und ausgetauscht. Alle Datenbanken, welche miteinander kommunizieren, müssen irgendwo registriert werden. Für jeden Journal Eintrag wird dann festgehalten, welche der registrierten Datenbanken den Journal Eintrag schon verarbeitet haben. Haben alle registrierten Datenbanken den Journal Eintrag verarbeitet, kann der Journal Eintrag gelöscht werden.

Das Programmieren einer proprietären Replikation lohnt sich nur, wenn ganz bestimmte Ziele damit erreicht werden sollen, beispielsweise möchte man die Datenmenge kontrollieren können, welche bei einer Replikation ausgetauscht werden, weil die Benutzer aus Hotelzimmern in Indonesien mit einem analogen Modem über schlechte Telefonleitungen mit der Zentrale in der Schweiz replizieren möchten.

Mögliche Prüfungsfragen

- Angenommen an einer Spezifikationssitzung wird vorgeschlagen, eine proprietäre Replikation zwischen den Datenbanken der europäischen Firmenniederlassungen ihrer Firma zu implementieren. Auf diese Art und Weise sollen die Daten schnell und sicher an alle Niederlassungen verteilt werden. Führen sie Vor- und Nachteile einer derartigen proprietären Replikation auf. Schlagen sie eine Alternative vor, falls sie mit einer derartigen Lösung nicht einverstanden sind.
- An der wöchentlichen Teamsitzung wird die proprietäre Replikation diskutiert, welche entwickelt werden soll. Unklar ist, ob die Einträge in die Journal Tabellen von der Applikation oder von Triggern geschrieben werden sollen. Führen sie Vor- und Nachteile der beiden Lösungen auf und entscheiden sie sich für eine Variante.

6.8 Die Möglichkeiten im Vergleich

Normale Tabelle Die Daten sind lokal vorhanden und können sofort gelesen werden. Schnell.

Normale View Der Zugriff auf die Daten kann pro View festgelegt werden, was feinere Unterscheidungen bei den Zugriffsrechten erlaubt. Alle SQL Statements werden von Oracle für die original Tabellen umgeschrieben. Performance suboptimal.

Materialized View für lokale Tabelle, Abgleich nach Bedarf Bei Änderungen müssen immer Originaltabelle und View zusammen nachgeführt werden. Deswegen ist die Performance bei Änderungen schlecht. Beim Lesen der Daten ist die Performance in beiden Fällen gut. Deswegen kann eine Materialized View dazu dienen, Aggregationen oder Joins im voraus zu berechnen.

Materialized View für lokale Tabelle, Abgleich periodisch Beim Lesen und beim Ändern der Daten in den original Tabellen ist die Performance gut. Bei den Materialized Views ist die Performance beim zwar auch gut, die Daten befinden sich aber nicht auf dem aktuellen Stand.

Materialized View für remote Tabelle, Abgleich nach Bedarf Die Performance ist bei Änderungen der original Tabellen sehr schlecht, weil die Materialized Views sofort und über das Netzwerk nachgeführt werden. Beim Lesen der Daten ist die Performance allerdings genauso gut wie bei den original Tabellen. Dies wird auch als *real-time snapshot* bezeichnet.

Materialized View für remote Tabelle, Abgleich periodisch Für Originaltabelle und Materialized View ist die Performance beim Lesen und beim Ändern der Daten gut. Bei den Materialized Views befinden sich die Daten aber nicht auf dem aktuellen Stand. Dies wird auch als *snapshot* bezeichnet.

DB Link Beim Lesen und beim Ändern der Daten in den original Tabellen ist die Performance gut. Der DB Link selber braucht fast keinen Platz. Da alle SQL Statements und alle Daten über das Netz übertragen werden müssen, ist die Performance auf der DB Link Seite besonders schlecht. Die Daten sind allerdings immer auf dem aktuellsten Stand.

Mögliche Prüfungsfragen

- Sie planen eine Applikation, welche vom Kunden an fünf verschiedenen Orten in der Schweiz eingesetzt wird. Es handelt sich um Verkaufsdaten, welche für Einkauf, Lagerhaltung, Transport, etc. verwendet werden. Was für eine Lösung (Web Applikation, verteilte Datenbanken mit DB Links zu einem zentralen Server, verteilte Datenbanken auf Notebooks mit Modem) empfehlen sie? Begründen sie ihre Wahl. Führen sie auf, welche Annahmen sie gemacht haben, um diese Frage zu beantworten.

6.9 Datafiles und Tablespaces

Damit die Datenbank wachsen kann und I/O Probleme möglichst einfach behoben werden können, gilt es folgende Ziele im Auge zu behalten:

- I/O Operationen sollten gleichmässig auf alle Platten verteilt sein.
- Das Hinzufügen von weiteren Platten muss einfach sein.

Um die gleichmässige Verteilung der I/O Operationen zu prüfen, verwendet man die V$FILESTAT View. Die dort referenzierten Dateien findet man in der V$DATAFILE View. Die letzten beiden Spalten der V$FILESTAT View werden nur gefüllt, wenn TIMED_STATISTICS aktiviert wurden.

Diese Daten zeigen nur an, wie der Oracle Prozess die Daten zu schreiben und zu lesen versucht. Es kann gut sein, dass diese *Zugriffe vom Betriebsystem auch wieder gecached* werden.

Falls *im SYSTEM Tabelspace zuviele Zugriffe* vorkommen, kann das auf folgende Probleme deuten:

- Shared Pool ist zu klein. Lösung: Shared Pool Grösser überprüfen.
- Daten Segmente werden in den SYSTEM Tablespace geschrieben (Default Tablespace für Benutzer nicht gesetzt?)
- Sortiersegmente werden in den SYSTEM Tabelspace geschrieben (Temporary Tablespace für Benutzer nicht gesetzt?)

6.9.1 Tablespaces, Default Konfiguration

Die einzelnen Datafiles sollte *nie grösser als 500MB* sein. Im Normalfall erzeugt man *pro Tablespace ein Datafile*. Wenn man dann für die Tabellen und die Indexe einer Applikation verschiedene Tablespaces anlegt, so werden Operationen, welche Daten in den Tabellen und im Index gleichzeitig ändern, weniger unter I/O Flaschenhälsen zu leiden haben.

Infos zu den Tablespaces gibt es in der Administrator View DBA_TABLESPACES. Was die richtige Aufteilung angeht:

SYSTEM Dieser Tablespace *enthält den Data Dictionary*. Da dies der einzig zwingend notwendige Tablespace ist, fungiert der SYSTEM Tablespace auch als default Tablespace für Tabellen, Indexe, Rollback Segmente und Temporäre Segmente, sofern sie nicht explizit in anderen Tablespaces angelegt werden. Im SYSTEM Tablespace sollten sich *keinerlei andere Datenbank Objekte* befinden! Typischerweise ca. 40 bis 60MB gross.

TEMP Der Tablespace für Temporäre Segmente wird vor allem für Sortier-Operationen verwendet. Die default Storage Parameters sind meist verschieden von denjenigen der anderen Tablespaces, siehe das Kapitel zur Sort Area Grösse.

Rollback Da Rollback Segmente (RBS) sehr aktiv sind, empfiehlt es sich, Rollback Tablespaces auf mehrere Platten zu verteilen.

USERS Hier gibt es Platz für alle privaten Tabellen und Indexe der Benutzer. Im Normalfall ist dieser Tablespace klein oder gar nicht vorhanden, da von den Mitarbeitern des Kunden nur die Applikationen die Datenbank verwenden; die Mitarbeiter sollten keinen direkten Zugang zur Datenbank haben.

xxx_DATEN Alle Tabellen der Applikation, eventuell verteilt auf mehrere Tablespaces, eventuell aufgeteilt nach Applikation. Bei einer Aufteilung nach Applikation kann man beim Ausserbetriebsetzen einer Applikation alle dazugehörigen Tablespaces ganz einfach offline nehmen. Desweiteren sollten Tabellen, welche oft gleichzeitig benutzt werden, auf verschiedene Tablespaces und auf verschiedene Platten aufgeteilt werden. Um die Belastung besser zu verteilen, sollten oft benutzte Tabellen mit weniger oft benutzten Tabellen gemeinsam abgelegt werden.

xxx_INDEX Alle Indexe, eventuell verteilt auf mehrere Tablespaces, eventuell aufgeteilt nach Applikation. Die Indexe sollten nicht auf derselben Platte wie die dazugehörigen Daten, damit Daten und Indexe möglichst gleichzeitig geändert werden können. Falls es eine Tabelle gibt, auf der besonders

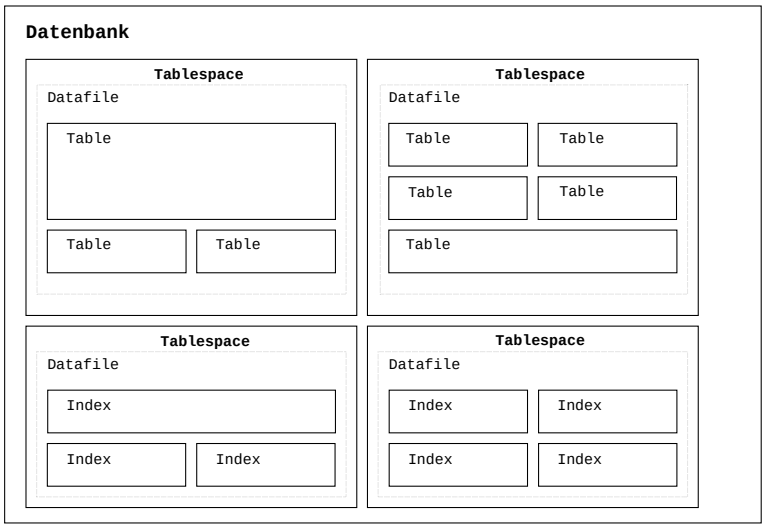

Abbildung 6.3: Indexe und Tabellen in getrennten Tablespaces

viele Inserts gemacht werden, könnte der dazugehörige Index auf einer eigenen Platten liegen.

PRODUCTS Platz für alle Datenbank Objekte, welche von Oracle Werkzeugen verwendet werden.

Diese Aufteilung hat den Vorteil, dass sie pro Tablespace eigene Storage Parameters angeben können, und sie können pro Tablespaces angeben, welche Datenfiles auf welchen Laufwerken verwendet werden sollen.

Mögliche Prüfungsfragen

- Warum sollten sie keine eigenen Objekte im SYSTEM Tablespace von Oracle anlegen?
- Angenommen sie haben eine Oracle Datenbank, auf der die Applikationen TRIMO und SUMO laufen. Die Benutzer dürfen keine eigenen Datenbank Objekte anlegen. Es sind keine Werkzeuge im Einsatz, welche eigene Daten auf der Datenbank speichern. Welche Tablespaces legen sie an? Geben sie jedem Tablespace einen Namen und beschreiben sie, welche Art von Daten darauf gespeichert werden. Begründen sie ihre Aufteilung.

6.9.2 Striping

Falls möglich, sollen alle Daten, welche gleichzeitig benötigt werden, auf verschiedenen Festplatten liegen. So können die Daten von mehreren Leseköpfen gleichzeitig gelesen werden. Deswegen werden *Tabellen und Indexe auf verschiedene Tablespaces* verteilt, und deswegen werden *oft benutzte Tabellen auf verschiedene Tablespaces* verteilt.

Es ist möglich, auch *eine Tabelle auf mehrere Platten* zu verteilen. Dies lohnt sich nur bei grossen Tabellen, welche von vielen Benutzern gleichzeitig benutzt werden. Die Aufteilung nennt man *Striping*.

Dies kann man *manuell* machen, indem man einem Tabellensegment ein neues Extent in einem anderen Datafile hinzufügt. Dies geschieht mit dem ALTER TABLE ... ALLOCATE EXTENT Befehl. Die neuen Datafiles müssen zum selben Tablespace gehören wie die bestehenden Datafiles.

Einfacher ist dies *automatisch* zu erledigen, und zwar durch folgenden Trick: Man legt einen Tablespace mit n Datafiles an, und gibt jedem dieser Datafiles eine bestimmte Grösse x. Dann legt man die Tabelle an, und gibt an, dass die Tabelle aus n Extents einer Grösse y bestehen soll, wobei y ein wenig kleiner als x ist.

Beispiel:

```
SQL> create tablespace prd_data datafile
       'c:\orant\database\prd1.dbf' size 200m,
       'd:\orant\database\prd2.dbf' size 200m,
       'e:\orant\database\prd3.dbf' size 200m,
       'f:\orant\database\prd4.dbf' size 200m,
     default storage (initial 20m, next 20m, pctincrease 0);

Tablespace created.

SQL> create table people (
       ...
       ... )
     storage (initial 199m, next 199m,
              minextents 4, pctincrease 0)
     tablespace prd_data;

Table created.
```

Die Tablespaces werden auf Datafiles aufgeteilt. Diese Aufteilung wird in der View DBA_DATA_FILES aufgeführt. Die Verteilung der Extents, welche in diesem Fall interessiert, wird in der view DBA_EXTENTS aufgeführt.

Mögliche Prüfungsfragen

- Worauf müssen sie achten, wenn sie einen Temporary Tablespace anlegen?
- Inwiefern wirkt Input/Output Performance limitierend auf eine Datenbank? Wie kann man dem entgegen wirken?
- Tablespaces bieten einen administrativen Vorteil und möglicherweise einen Performance Vorteil. Erklären sie.
- RAID (Redundant Array of Inexpensive Disks) Systeme bieten einerseits Datensicherheit, können aber auch die Performance steigern. Beschreiben sie eine RAID Architektur und zeigen sie, warum die Performance steigt. Erklären sie kurz, wie man die Datenbank aufsetzen müsste, wenn kein RAID zur Verfügung steht.
- Angenommen sie haben eine Produktionsanlage, mit der ein programmierbarer Stromzähler hergestellt wird. Für jedes Gerät speichern sie während der Produktion in der Datenbank tausende von Datensätzen in die Tabelle TEST_DATEN. Nun können damit bei späteren Reklamationen genau

herausfinden, wie das Gerät damals programmiert worden ist. Im Laufe der Zeit geht der Plattenplatz langsam zuneige und sie kaufen eine neue Platte. Wie verwenden sie die neue Platte? Zeichnen sie ein kleines Diagramm, welches den Zusammenhang zwischen der Tabelle TEST_DATEN und den Festplatten zeigt. Beschriften sie alle Oracle Speicherstrukturen, welche sie im Diagramm verwendet haben. Das Diagramm muss detailliert genug sein, damit einem DBA klar ist, welche SQL Befehle auszuführen sind. Die SQL Befehle selber sind irrelevant.

Index