DOAG EDITION PRAXISWISSEN

Oracle Tuning in der Praxis

Rezepte und Anleitungen für Datenbankadministratoren und -entwickler

2. Auflage

Frank Haas

HANSER

Inhalt

1 Oracle-Design

Thema des ersten Kapitels ist, welche Möglichkeiten Oracle Ihnen für ein Design, das eine gute Performance gewährleistet, zur Verfügung stellt.

Performance ist idealer weise nichts, was man nachträglich auf die Datenbank aufpfropft, sondern sie sollte von Anfang an im Design berücksichtigt werden. Das setzt natürlich voraus, dass Sie bereits beim Design der Applikation mit den Möglichkeiten, die das jeweilige Datenbanksystem anbietet, vertraut sind.

Unabhängig davon, welchen Typ von Objekten Sie verwenden – wobei es sich zum großen Teil um Tabellen und Indizes handeln wird, müssen Sie diese Objekte physikalisch abspeichern. Mit der physikalischen Organisation treffen Sie auch Entscheidungen, die sich auf die Performance auswirken. Dieses Thema wird noch in einem separaten Kapitel im Detail besprochen.

Während sich der Rest des Buches vornehmlich der Untersuchung und Analyse bestehender Applikationen widmet, behandelt dieses Kapitel, wie Sie Applikationen bauen, die bereits im Design Performance berücksichtigen. Dabei liegt der Schwerpunkt auf der Datenbank und SQL. Gleichzeitig ist es wohl auch das Kapitel des Buches, das sich am stärksten an den Entwickler richtet.

1.1 Datenhaltung in Oracle

1.1.1 Tabellen

Für die Speicherung der applikatorischen Daten bietet Oracle verschiedene Möglichkeiten an. Im Regelfall werden Sie ganz normale relationale Tabellen verwenden. Es existieren zwar auch Objekttypen, aber diese sollten Sie nicht zur Speicherung verwenden, weil dort zuviel hinter der Bühne passiert. Wenn Sie beispielsweise eine Nested Table oder eine Object Table anlegen, dann generiert Oracle im Hintergrund versteckte Spalten, die Sie der Tabelle erst mal nicht ansehen. Für die Applikationsentwicklung sind diese Typen exzel-

lent geeignet, nicht aber für die Persistenz. Im Regelfall sollten Sie ganz normale relationale Tabellen verwenden. Die einfache CREATE TABLE-Anweisung erzeugt eine relationale Tabelle, genauer gesagt, eine Heap-Tabelle. Ein Heap ist in der konventionellen Programmierung eine bekannte Struktur. Eine Heap-Tabelle hat keine eingebaute Reihenfolge der Einträge, letztere können dynamisch hinzugefügt und gelöscht werden. 95% oder mehr der Tabellen in Ihrer Applikation werden solche Tabellen sein. Theoretisch könnten Sie auch beim Anlegen der Tabelle angeben, dass es sich um eine Heap-Tabelle handelt: CREATE TABLE .. ORGANIZATION HEAP. Aber da das ohnehin die Voreinstellung ist, macht (und braucht) das niemand. Weitere Details finden Sie in der offiziellen Oracle-Dokumentation in [OraCon 2005] und [OraSql 2005].

Temporäre Tabellen

Es gibt auch temporäre Tabellen, die zur Speicherung von Zwischenresultaten verwendet werden können, diese werden mit CREATE GLOBAL TEMPORARY TABLE erzeugt. Diese Tabelle ist dann für alle Sessions, also global innerhalb der Datenbank, sichtbar. Im Bereich Performance ist der Unterschied zwischen temporären und regulären Tabellen aber oft nicht so signifikant. Betrachten Sie temporäre Tabellen also vor allem als applikatorische Möglichkeit, nicht als Mittel zur Steigerung der Performance. Zwar generiert eine temporäre Tabelle kein Redo. Das ist gut für die Performance. Bei mehrstufigen Abfragen – also Query A ist Input für Query B und B ist potenziell wieder Input für eine neue Query etc. – sind sie sicher eine sehr naheliegende Lösung. Aber Sie haben – teilweise versionsbedingt – auch keine Statistiken, was wiederum unter Umständen einen schlechten Zugriffsplan verursacht. Allerdings lässt sich dieses Problem dann durch manuelles Setzen der Statistiken über DBMS_STATS oder das dynamische Sammeln der Statistiken (ab 9.2) lösen. Bei einer temporären Tabelle müssen Sie angeben, ob die Daten nur für die Transaktion oder für die ganze Session gültig sein sollen. Dies erfolgt über die ON COMMIT-Klausel: ON COMMIT DELETE ROWS löscht die Daten nach jedem COMMIT, das ist auch die Voreinstellung. Bei ON COMMIT PRESERVE ROWS bleiben die Daten während der ganzen Session erhalten. Temporäre Tabellen können indiziert werden, aber Fremdschlüssel sind nicht erlaubt. Temporäre Tabellen sind immer relationale Tabellen. Temporäre Tabellen können auch nicht parallelisiert werden. Sehr gut geeignet sind temporäre Tabellen für die Speicherung von Zwischenergebnissen oder Abfragen auf die dynamischen Performance Views, die V$-Views. Die V$-Views werden uns später noch öfters begegnen. Das sind alles Views, deren Name mit V$ beginnt. Sie enthalten die aktuellen Betriebsdaten der Datenbank. So erfahren Sie über V$SESSION, welche Benutzer gerade in der Datenbank angemeldet sind, V$PROCESS zeigt Ihnen die Prozesse der Datenbank, V$INSTANCE zeigt Ihnen, seit wann die Datenbank läuft, und so weiter und so fort. Diese V$-Views sind aber keine „normalen“ Views, sondern Strukturen, die in der Datenbank eingebaut sind. Das kann dazu führen, dass Abfragen auf diese Views, die Sie mit Abfragen auf normale Tabellen joinen, ungültige oder falsche Ergebnisse bringen. Um dieses Problem zu vermeiden, müssen Sie zuerst die Daten der V$-View in eine Zwischentabelle übertragen.

Externe Tabellen

Externe Tabellen wurden mit Oracle 9*i* eingeführt. Eine externe Tabelle ist eine Tabelle, die in Form einer Datei auf dem Betriebssystem vorliegt. Die Datei ist dann nur lesbar. Das spart zum einen Platz, weil man die Tabelle dann ja nicht noch auch in die Oracle-Datenbank laden muss. Zum andern spart man auch Zeit, die Zeit für das Laden der Daten entfällt ja auch. In Oracle-Versionen vor 9*i* können Sie dasselbe mit BFILEs erreichen. Dabei folgt das Format der Dateien den Formaten, wie sie im SQL*Loader verwendet werden:

```
CREATE TABLE dept_external
(deptno number(2), dname varchar2(14), loc varchar2(13))
ORGANIZATION EXTERNAL
(TYPE oracle_loader
DEFAULT DIRECTORY DATA_DIR
ACCESS PARAMETERS (
records delimited by newline characterset WE8ISO8859P1
badfile ' dept_external.bad'
logfile 'dept_external.log'
fields terminated by "," optionally enclosed by '"' ldrtrim
reject rows with all null fields (
deptno char(25),
dname char(25),
loc char(25))
LOCATION ('dept_external.ctl'))
REJECT LIMIT UNLIMITED;
```

Berücksichtigen Sie externe Tabellen besonders im Data Warehouse-Bereich, wenn verschiedene Daten auf und von unterschiedlichen Systemen miteinander abgeglichen werden müssen.

Seit Version 10*g* können externe Tabellen auch geschrieben werden, das erfolgt über den ORACLE_DATAPUMP-Treiber. Die entstandene Datei ist aber nicht im ASCII-Format; auch hier ein Beispiel:

```
CREATE TABLE dept_external_unload O
RGANIZATION EXTERNAL
(TYPE oracle_datapump
DEFAULT DIRECTORY TEMP_DIR
LOCATION ('dept_external_unload.ctl'))
AS Select * from dept;
```

Primärschlüssel

Ein Primärschlüssel definiert eindeutig jeden Datensatz in einer Tabelle und verhindert somit Duplikate. Sie können als Primärschlüssel eigentlich alles verwenden, was die Row eindeutig definiert. Dabei kann das Schlüsselfeld natürlich vorkommende Werte annehmen – sofern es solche in der Tabelle gibt, oder man nimmt ein eigens erzeugtes Schlüsselfeld. Letzteres ist vorteilhafter, da es die Applikationslogik von der technischen Implementierung abkoppelt. Sie vermeiden dadurch Probleme, falls aus irgendwelchen Gründen das Schlüsselfeld verändert werden muss. Dies sollte zwar in der Theorie nicht vorkommen, tut es in der Praxis dann aber doch. Mit einem technischen Schlüssel ist das und manches andere leichter. Steht die Tabelle in einer hierarchischen Beziehung, sind auch zusammengesetzte Primärschlüssel interessant. Nehmen wir als Beispiel die zwei Tabellen Rech-

nungskopf und Rechnungsdaten. Im Rechnungskopf sind dann die allgemeinen (einmaligen) Daten wie Kunde, Zahlungsziel, Gesamtsumme, Kontoverbindung etc. abgespeichert, während in den Rechnungsdaten die zugeordneten Rechnungsposten abgespeichert werden. Als Schlüssel im Rechnungskopf wird eine fortlaufenden eindeutige Rechnungsnummer verwendet. Das Schlüsselfeld in der Tabelle Rechnungsdaten wird dann aus dem Schlüsselfeld im Rechnungskopf und einem weiteren Feld, das den zugeordneten Rechnungsposten eindeutig identifiziert, gebildet.

Generell sollte also jede Tabelle in der Datenbank zumindest einen Primärschlüssel haben. Oracle realisiert den Primärschlüssel dann über einen eindeutigen B*-Baum-Index. Sie sind nicht verpflichtet, einen Primärschlüssel festzulegen. Manchmal scheint es auch nicht unbedingt notwendig. Extra einen Index anlegen für eine Tabelle mit nur 100 Datensätzen? Legen Sie aber trotzdem einen Primärschlüssel an. Das ist eine Investition in die Zukunft. Angenommen, in einem Jahr wird beschlossen, dass noch eine Standby-Datenbank neben der Produktion aufgebaut werden soll. So weit, so schön, aber: Tabellen, die in Standby-Umgebungen repliziert werden sollen, müssen einen Primärschlüssel haben. Bedeutet also konkret, dass Sie Ihre Applikation einem Redesign unterziehen müssen. Das wäre aber nicht notwendig gewesen, wenn die Tabellen gleich von Anfang an mit Primärschlüsseln versehen worden wären. Abgesehen davon geben Sie dem Optimizer damit mehr Informationen, und je mehr Informationen der Optimizer hat, desto bessere Zugriffspläne kann er erzeugen.

Bei künstlichen Schlüsselfeldern empfehlen sich fortlaufende Nummern. Mit Sequenzen und Triggern lassen sich die Schlüsselfelder dann fast automatisch nachführen, das sehen wir später dann noch im Detail. Beherzigen Sie die Empfehlungen, die im Kapitel 6 von [OraAppDev 2005] für Primärschlüssel gegeben werden:

- Verwenden Sie eine Spalte, deren Werte eineindeutig sind. Ein Primärschlüssel dient dazu, eine Zeile eindeutig zu identifizieren.
- Verwenden Sie eine Spalte, deren Wert später nicht mehr verändert wird. Der Primärschlüssel sollte lediglich dazu dienen, den Datensatz eindeutig zu kennzeichnen. Er sollte keine andere (zusätzliche) Bedeutung haben.
- Verwenden Sie eine Spalte, die als NOT NULL deklariert werden kann. NULL-Werte sind in Primärschlüsseln nicht zulässig.
- Verwenden Sie einen numerischen Datentyp, dann können die Werte für den Primärschlüssel über eine Sequenz bereitgestellt werden.
- Vermeiden Sie Primärschlüssel, die aus mehreren Feldern zusammengesetzt werden; die Wartung ist in diesem Fall wesentlich aufwändiger.

Bei der Spezifikation der Primärschlüsselspalte ist NOT NULL nicht zwingend, aber sehr zu empfehlen, wie das folgende Beispiel anschaulich zeigt:

```
SQL> create table pk_example (x number primary key);
Table created.

SQL> insert into pk_example values (1);
1 row created.
```

```
SQL> insert into pk_example values (null);

insert into pk_example values (null)
                               *

ERROR at line 1:
ORA-01400: cannot insert NULL into ("SYSTEM"."PK_EXAMPLE"."X")

SQL> commit;

Commit complete.

SQL> select * from pk_example;

         X
----------
         1
```

Die Implementation des Primärschlüssels erfolgt hier über einen eindeutigen Index. Das ist aber nicht zwingend. Existieren bereits Indizes, schaut Oracle zuerst, ob der Primärschlüssel über diese bestehenden Indizes realisiert werden kann. Diese Indizes müssen nicht einmal eindeutig sein.

Wie Sie hier sehen, können Sie einen NULL-Wert nicht eingeben; also ist es auch nicht sinnvoll die Spalte so zu deklarieren, es gibt ja einen Fehler, wenn Sie einen NULL-Wert einfügen wollen.

Das ist übrigens nicht das Gleiche wie ein eindeutiger Index auf dieser Spalte, wie man hier sieht:

```
SQL> create table unique_example (x number);
Table created.

SQL>  create unique index uniq_ix on unique_example (x);
Index created.

SQL> insert into unique_example values (1);
1 row created.

SQL> insert into unique_example values (null);
1 row created.

SQL> commit;
Commit complete.
```

Sie können also in diesem Fall trotz des eindeutigen Index NULL-Werte eingeben. Es kommt aber noch besser, das lässt sich auch (beliebig oft) wiederholen:

```
SQL> insert into unique_example values (null);
1 row created.

SQL>  commit;

Commit complete.

SQL> select * from unique_example;

         X
----------
         1


3 rows selected.
```

Register

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W