Oracle Index Tuning &Admin

Marco Patzwahl
MuniQSoft GmbH
München-Unterhaching

Schlüsselworte:

SQL, PL/SQL, DBA

Zusammenfassung

Indizes sind ein erprobtes Mittel, um SQL-Abfragen zu beschleunigen. Aber haben sie auch Nachteile? Welche Performance geht durch Indizes verloren und welche Indizes sind die schnellsten?

Einleitung

Ein Index indiziert eine oder mehrere Spalten einer Tabelle. Bei Abfragen auf indizierte Spalten werden die dazu passenden Zeilen schneller gefunden. Bei Inserts/Updates/Deletes auf indizierten Spalten wird die Transaktion jedoch verlangsamt!

Vorteile von Indizes:

- u.U. schneller Zugriff bei Select, Update und Delete
- Sortierungen schneller
- COUNT(*) schneller bei Bitmap Index oder indizierten NOT NULL Spalten

Nachteile von Indizes

- Längere Verarbeitungszeit bei Insert/Update/Delete
- Höherer Pflegeaufwand durch Index (Resourcenverbrauch)
- Index-Blöcke können Cache dicht machen (und Tabellen-Blöcke aus dem Cache vertreiben)

Indexerstellung
Ein Index kann auf jede beliebige Spalte einer Tabelle (Ausnahme Long und Lob) gesetzt werden. Einspaltige oder mehrspaltige Indizes sind möglich.

Syntax:

```
CREATE [UNIQUE] INDEX [<owner>.]<index_name>
ON [<owner>.]<tab_name>
(<spalten_name> [ASC|DESC],
<spalten_name> [ASC|DESC], ...)
[LOGGING|NOLOGGING] [ONLINE]
[TABLESPACE <tablespace_name>]
[COMPRESS <int> | NOCOMPRESS]
[NOSORT]
[PARALLEL <int> |NOPARALLEL]
[PCTFREE <int>] [INITRANS <int>];
```

Besonderheiten des Index im Vergleich zur Tabelle:

- Es gibt keinen Default Index Tablespace. Ein Index landet (ohne Angabe) im Default Tablespace des Benutzers
- PCTFREE hält bei Indizes den Platz für Updates und INSERTS frei!
- Es gibt keinen internen Update auf einen Index-Wert. Stattdessen wird der Wert gelöscht und neu eingefügt
- Ein Indexeintrag darf maximal 78% eines Blocks betragen, sonst bekommt man den Fehler (8k Blockgröße): `SQL-Fehler: ORA-01450: Maximale Schlüssellänge (6398) überschritten`
- Maximal 32 Spalten (Bitmap Index: 30) lassen sich von einem Index erfassen

Bitmap Index

Voraussetzungen:

- Enterprise oder Personal Edition
- Bei Tabellen mit geringer Insert und Update Aktivität auf indizierte Spalten (Data Warehouse)

Vorteile:

- Bei größeren Datenmengen mit geringer Kardinalität (z. B. 80 Mio. Einwohner, Bitmap Index auf Spalte Bundesländer (16))
- Schneller, wenn Abfragen aus mehreren WHERE- Bedingungen bestehen, die mit OR oder AND verknüpft sind
- Bei Read-Only Tabellen besser
- Benötigen weniger Speicherplatz als B*Indizes, wenn Kardinalität klein (teilweise nur 2% von B*Indizes)
- Count Funktion ist schneller

Nachteile:

- INSERT, UPDATE und DELETE dauern länger. Da beim Ändern eines Wertes alle Zeilen mit diesem Wert gesperrt werden, kann es leichter zu Deadlocks kommen
- Index-Werte werden komprimiert gespeichert. Bei Änderungen muss zuerst dekomprimiert, danach manipuliert, der Index-Zweig neu aufgebaut und wieder komprimiert werden

Syntax (verkürzt):

```
CREATE BITMAP INDEX [<owner>.]<index_name>
ON [<owner>.]<tab_name>
(<col_name> [ASC|DESC], <col_name> [ASC|DESC] ...)
[TABLESPACE <tbs_name>];
```

Beispiel:

```
CREATE TABLE scott.bm_test AS SELECT rownum
nr,decode(floor(DBMS_RANDOM.value(0,4)),0,
'Nord',1,'Süd',2,'Ost',3,'West') region
 FROM dba_objects; -- auf 1 Mio Zeilen

CREATE BITMAP INDEX scott.bm_ind
ON scott.bm_test(region) TABLESPACE indx;
```

```
BEGIN dbms_stats.gather_index_stats(
ownname=>'SCOTT',indname=>'BM_IND');
END;
/
SELECT blevel, leaf_blocks, distinct_keys,
       avg_leaf_blocks_per_key FROM DBA_INDEXES
WHERE index_name='BM_IND';
=> 1,170,4,42    (Bitmap Index)
=> 2,4440,4,1110 (Für Nonuniqe Index, >Faktor 26 )
```

Der Bitmap Index ist also Faktor 26 kleiner als ein normaler B-Tree Index.

Reverse Key Index

Beschreibung:

- Ein Reverse Key Index indiziert die Werte rückwärts
- So wird z.B. aus 123 der Wert 321

Vorteile:

- Bei Primary Key Erzeugung durch Sequenzen sind Inserts schneller
- Verteilt die Index-Einträge auf verschiedene Index-Blöcke und damit gibt es weniger Sperrkonflikte
- Vorteilhaft in einer RAC-Umgebung

Nachteile:

- Keine Index Range Scans möglich

Syntax:

```
CREATE INDEX [<owner>.]<index_name>
ON [<owner>.]<tab_name>
(<col_name> [ASC|DESC], <col_name> [ASC|DESC] ...)
[TABLESPACE <tbs_name>] REVERSE;
```

Beispiele:

```
CREATE INDEX scott.brd_ri
ON scott.telefonbuch  (kunden_id)
TABLESPACE index_tbs REVERSE;
```

Index-Tuning

Index wird bei Primary Key/ Unique Key automatisch angelegt.
Es ist günstiger, zuerst den Unique Index und dann den Constraint auf diese Spalte anzulegen.
NOLOGGING schreibt nur einen geringen Bruchteil der Informationen in die Redologdatei. Im Fall eines Server-Crashs, wird der Index jedoch unbrauchbar.

```
CREATE TABLE t (id NUMBER);

CREATE UNIQUE INDEX i ON t(id)
TABLESPACE indx_tbs NOLOGGING;
```

```
ALTER TABLE t ADD CONSTRAINT pk PRIMARY KEY(id);
```

Alternativ kann wenigstens der Tablespace des Index bei Erzeugen des Constraints angegeben werden:

```
CREATE TABLE t (id NUMBER CONSTRAINT pk
PRIMARY KEY USING INDEX TABLESPACE ts_idx);
```

Achtung: Wenn der Index eigenständig angelegt wurdem bleibt er Drop auf den Primary Key bestehen. IM anderen Falle wird er mitgelöscht.

Index Tuning durch Selektion

Sie können entscheiden, dass nur bestimmte Werte indiziert werden sollen. Nur nach diesen Werten kann dann natürlich über den Index gesucht werden.
Beispiel: Nur Münchner Telefonnummern sollen indiziert werden:

```
CREATE INDEX scott.brd_tel_ix ON scott.brd(
CASE WHEN vorwahl='089' THEN vorwahl ELSE NULL END);
```

Originalgröße des Index (ohne Selektion): 2GB
Größe des Index (mit Selektion): 16MB

SELECT dazu (der den Index dann auch benutzt):

```
SELECT * FROM scott.brd WHERE (CASE WHEN vorwahl='089' THEN vorwahl
ELSE NULL END)='089'
```

Compress Option

Mit der COMPRESS Option können führende Index-Spalten, die viele gleiche Werte besitzen, platzsparend im Index untergebracht werden:

```
CREATE INDEX scott.i
ON scott.emp(deptno,job) COMPRESS 2;
```

Sie können vom bestehenden Index eine Analyse erzeugen, um zu sehen, wie viel Prozent ein COMPRESS einsparen würde:

```
ANALYZE INDEX scott.i VALIDATE STRUCTURE;
SELECT opt_cmpr_count,opt_cmpr_pctsave
FROM index_stats;
```

Nutzen der Compression:
Hier werden gleiche Index Einträge komprimiert.

```
CREATE TABLE t (x VARCHAR2(100),y VARCHAR2(100));
INSERT INTO t SELECT object_type,status FROM dba_objects;
CREATE INDEX i ON t(x,y) COMPRESS 2;
```

Folgende Indizes wurden erzeugt:

```
i(x,y)           100 Leaf Blocks
i(x)+i(y)        77+66 = 143 Leaf Blocks
i(x,y) compress 2 43 Leaf Blocks
i(x,y) compress 1 67 Leaf Blocks
```

Index-Größe abschätzen:
Mit folgendem Skript lässt sich die Größe (hier in MB) abschätzen, die ein Index bei einer Erstellung benötigen würde.
Hinweis: Compress wird bei Indexberechnung nicht berücksichtigt.

```
DECLARE
 ub  NUMBER;
 ab  NUMBER;
 s CLOB;
BEGIN
  s:='CREATE INDEX x.x ON scott.bug ("TEXTID", "LOAD", "GEO")
  TABLESPACE USERS';
  dbms_space.create_index_cost(s, ub, ab);
  dbms_output.put_line('Used MB: ' ||round(ub/1024/1024,2);
  dbms_output.put_line('Alloc MB:' ||round(ab/1024/1024,2);
END;
/
```

Index-Reorg (1)
Der Platz in den Index-Blöcken wird nach einem Delete u.U. nicht mehr verwendet.
Indizes wachsen nach vielen Deletes und anschließenden Inserts stark an und sollten häufiger reorganisiert werden.

```
ANALYZE INDEX <owner>.<indx> VALIDATE STRUCTURE;
SELECT name, del_lf_rows, lf_rows - del_lf_rows lf_rows_used,
to_char(del_lf_rows / (lf_rows)*100,'999.99999') ratio,
OPT_CMPR_COUNT,OPT_CMPR_PCTSAVE
FROM index_stats where name = upper('<indx>');
```

Wenn die Anzahl der gelöschten Zeilen 10 - 15% beträgt, sollte der Index reorganisiert werden (Metalink Note 30405.1):

```
ALTER INDEX <owner>.<indx> REBUILD ONLINE;
```

Index-Reorg (2)
Eine andere Möglichkeit ist, das Package dbms_space anzuwenden. Das Package kann die Freelist der Index-Blöcke auslesen und damit ihren Füllpegel in den folgenden Bereichen bestimmen:
<25%
<=50%
<=75%
<=100%
Ein Skript mit einer Pipelined Funktion, die diese Analyse durchführt, ist beim Referenten erhältlich.

Index-Statistiken
Sie sollten Index Statistiken nur mit dem Package DBMS_STATS (.GATHER_INDEX_STATS, .GATHER_SCHEMA_STATS oder .GATHER_DATABASE_STATS) erzeugen
Die folgenden Aufrufe werden nicht mehr empfohlen:

- `ANALYZE INDEX idx COMPUTE STATISTICS;`
- `ANALYZE INDEX idx ESTIMATE STATISTICS …;`
- `CREATE INDEX idx ON t(c) COMPUTE STATISTICS;`

- `ALTER INDEX idx REBUILD COMPUTE STATISTICS;`

Bereits seit 10g werden Statistiken bei einem CREATE INDEX oder ALTER INDEX Befehl automatisch erzeugt!

Ergebnisse der Performance-Messungen:

- NOLOGGING beschleunigt die Index-Erstellung, birgt aber Gefahren
- Parallelisierung bringt nicht zwingend etwas
- COMPRESS kostet bei der Erstellung sogar weniger Zeit und man erhält einen schlanken Index
- ONLINE (nur in der EE) kostet keine zusätzlich Zeit, die Tabelle ist aber bereits während der Indexerstellung verfügbar
- Update und Delete sind ca. Faktor 4 langsamer. Ein Insert ist Faktor 14 langsamer!!!

Kontaktadresse:

Marco Patzwahl
MuniQSoft GmbH
Grünwalder Weg 13 a
D-82008 Unterhaching

Telefon: +49(0) 89-6228 6789-0
Fax: +49(0)89-6228 6789-50
E-Mail m.patzwahl@muniqsoft.de
Internet: http://www.muniqsoft.de