ORACLE INDEX SIKIŞTIRMA TEKNİKLERİ

İçindekiler

1. INDEX SIKIŞTIRMA

Daha önce hazırladığım bir yazıda (*Oracle Veritabanında Tablo ve Index Sıkıştırma*) tablo ve index sıkıştırmaya değinmiştim. Bu yazıda Index sıkıştırmanın inceliklerine değineceğim.

Create Index dediğimiz zaman yarattığımız Index'ler B-tree index'lerdir. Çalışmaların çoğu bu tarz index'ler üzerinden yapılır. Sıkıştırma konusunda da temel uygulamamız B-tree index üzerine olacak.

B-tree index'leri sıkıştırırken dikkat etmemiz gereken hangi sütunların tekrar ettiğidir. Örneğin elimizde 5 sütunu olan bir tablo olsun. Bunun üç sütunundan bir index oluşturacağımızı varsayalım. Index'e girecek sütunların değerlerinin aşağıdaki gibi olduğunu düşünelim:

```
ID         VERI            SAYI      TIP_1      TIP_2
=================================================
1          DENEME           1.333      A          -
1          DENEME           3.143      F          -
2          DENEME           5.511      G          -
1          DENEME           9.681      E          T
...
```

Bu tabloda **ID** ve **VERI** genelde aynı değerleri alırken; sayı farklı değerler almaktadır. Bunları bir üçlü olarak index'lerseniz, sıkıştırma işleminde bir bütün olarak (örneğin; *1–DENEME–1.333* şeklinde...) düşünmeniz gerekir. Eğer üçlü olarak sıkıştırmaya kalkarsak, sıkıştırabileceğimiz eş sayısı azalır. Ama bunu ikiliye düşürürsek, sıkıştırabileceğimiz eş sayısı ciddi oranda artar. **SAYI** sütunu index'in hâlâ bir parçasıdır ama sıkıştırılacak grubun içine alınmaz.

Bu bahsettiğim durumu index compress'ine ek bilgi girerek deneyebilirsiniz. Aşağıda bununla ilgili çeşitli testler yapılmıştır.

2. INDEX SIKIŞTIRMA TESTLERİ

Denemeleri yapabilmek için ön hazırlık olarak bir tablo yaratılmış ve buna 50bin kayıtlık değer girilmiştir.

```
CREATE TABLE D_CCEBI.DENEME
( ID NUMBER(10), VERI CHAR(1000), SAYI FLOAT(5) )
PCTUSED    0
PCTFREE    0
```

```
INITRANS   1
MAXTRANS   255
STORAGE    (
            INITIAL          30M
            MAXEXTENTS       UNLIMITED
            PCTINCREASE      0
            BUFFER_POOL      DEFAULT
           )
NOLOGGING
NOCOMPRESS
PARALLEL (DEGREE 4)
TABLESPACE TS_CCEBI;

set timing on
begin
    FOR i IN 1..50000
    LOOP
         insert into D_CCEBI.DENEME values( mod(i,10), 'Lorem ipsum dolor
sit amet, consectetur adipiscing elit.', DBMS_RANDOM.VALUE( )*10 );
        if mod(i,10000)=0 then
            COMMIT;
        END IF;
    END LOOP;
    COMMIT;
END;
PL/SQL procedure successfully completed.
Elapsed: 00:00:39.64
```

Farkedeceğiniz gibi tablonun ilk ve ikinci sütunu sıkça tekrar edecek değerlere sahiptir. Üçüncü ise 1 ile 10 arasında herhangi bir değer alabilecek rakamsal bir ifadedir. Sıkıştırılma işlemi kasıtlı olarak yapılmamıştır. Boyut olarak 57MB alan kaplamaktadır.

```
SELECT ROUND(bytes/1024/1024,2) TABLE_SIZE FROM DBA_SEGMENTS
WHERE OWNER='D_CCEBI' AND SEGMENT_NAME='DENEME';
TABLE_SIZE
----------
      57MB
```

2.1. Sıkıştırılmamış B-Tree Index

Şimdi bu tabloya hiç sıkıştırılmamış bir b-tree index yaratalım.

```
SET TIMING ON;
CREATE INDEX D_CCEBI.IDX_NOCOMPRESS ON D_CCEBI.DENEME( ID, VERI, SAYI )
NOLOGGING
TABLESPACE TS_CCEBI
PARALLEL (DEGREE 4);
Index created.
Elapsed: 00:01:17.67


SELECT ROUND(bytes/1024/1024,2) INDEX_SIZE FROM DBA_SEGMENTS
WHERE OWNER='D_CCEBI' AND SEGMENT_NAME='IDX_NOCOMPRESS';
```

```
INDEX_SIZE
----------
   66,25MB
```

Tabloya sorgu denemesi yaparsanız, Index'i kullanmak yerine full gittiğini görebilirsiniz. Ancak biz index kullanmasını zorluyoruz ve cost 1016 çıkıyor:

```
SELECT /*+INDEX( DENEME ) */ * FROM D_CCEBI.DENEME
WHERE ID = 5 AND VERI LIKE 'Lorem%' AND SAYI > 1;

-------------------------------------------------------------------------------------
| Id  | Operation        | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT |                |  6268 |  6292K|  1016   (1)| 00:00:13 |
|*  1 |  INDEX RANGE SCAN| IDX_NOCOMPRESS |  6268 |  6292K|  1016   (1)| 00:00:13 |
-------------------------------------------------------------------------------------
```

2.2. Tek Sütunu Sıkıştırılmış B-Tree Index

İkinci denemede sadece ID sütununu compress ederek, deneme yapacağız:

```
SET TIMING ON;
CREATE INDEX D_CCEBI.IDX_COMPRESS_COL1 ON D_CCEBI.DENEME( ID, VERI, SAYI )
COMPRESS 1
NOLOGGING
TABLESPACE TS_CCEBI
PARALLEL (DEGREE 4);
Index created.
Elapsed: 00:01:00.23

SELECT ROUND(bytes/1024/1024,2) INDEX_SIZE FROM DBA_SEGMENTS
WHERE OWNER='D_CCEBI' AND SEGMENT_NAME='IDX_COMPRESS_COL1';

INDEX_SIZE
----------
   65,31MB

-------------------------------------------------------------------------------------
| Id  | Operation        | Name              | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT |                   |  6268 |  6292K|  1002   (1)| 00:00:13 |
|*  1 |  INDEX RANGE SCAN| IDX_COMPRESS_COL1 |  6268 |  6292K|  1002   (1)| 00:00:13 |
-------------------------------------------------------------------------------------
```

Boyut olarak 1MB azalma var. Cost ise 14 puan düştü. Tabloya full gitmek hâlâ daha iyi. Kayda değer bir gelişme yok.

2.3. İki Sütunu Sıkıştırılmış B-Tree Index

Üçüncü denemede sadece ID ve VERI sütunlarını sıkıştırmasını istiyoruz:

```
SET TIMING ON;
```

```
CREATE INDEX D_CCEBI.IDX_COMPRESS_COL2 ON D_CCEBI.DENEME( ID, VERI, SAYI )
COMPRESS 2
NOLOGGING
TABLESPACE TS_CCEBI
PARALLEL (DEGREE 4);
Index created.
Elapsed: 00:00:04.36

SELECT ROUND(bytes/1024/1024,2) INDEX_SIZE FROM DBA_SEGMENTS
WHERE OWNER='D_CCEBI' AND SEGMENT_NAME='IDX_COMPRESS_COL2';

INDEX_SIZE
----------
    1,38MB

-------------------------------------------------------------------------------------------
| Id  | Operation        | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT |                  |  6268 |  6292K|    24   (0)| 00:00:01 |
|*  1 |  INDEX RANGE SCAN| IDX_COMPRESS_COL2 |  6268 |  6292K|    24   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

Index'in yaratılması için gereken süre %95 kısaldı; index boyutu %97 azaldı; cost %98 düştü. Ayrıca "*Index kullan*" zorlaması olmasa bile; tabloya full gitmek artık tercih edilmiyor.

2.4. Üç Sütunu Sıkıştırılmış B-Tree Index

Dördüncü denemede bütün sütunları sıkıştırmasını istiyoruz:

```
SET TIMING ON;
CREATE INDEX D_CCEBI.IDX_COMPRESS_COL3 ON D_CCEBI.DENEME( ID, VERI, SAYI )
COMPRESS 3
NOLOGGING
TABLESPACE TS_CCEBI
PARALLEL (DEGREE 4);
Index created.
Elapsed: 00:00:07.92


SELECT ROUND(bytes/1024/1024,2) INDEX_SIZE FROM DBA_SEGMENTS
WHERE OWNER='D_CCEBI' AND SEGMENT_NAME='IDX_COMPRESS_COL3';

INDEX_SIZE
----------
     4,5MB


-------------------------------------------------------------------------------------------
| Id  | Operation        | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT |                  |  6268 |  6292K|    73   (0)| 00:00:01 |
|*  1 |  INDEX RANGE SCAN| IDX_COMPRESS_COL3 |  6268 |  6292K|    73   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

Üçüncü adımdaki başarıyı elde edemedik. Bu beklenen bir durum. Çünkü aynı blok içinde ***ID*** ve ***VERI*** ikilisinin aynı olduğu birden fazla satır olacaktır. Ama bu ikiliye bir de ***SAYI*** değerini eklersek, tekrar sayısı düşer. Çünkü ***SAYI***, random bir değer tutmaktadır.

2.5. Varsayılan Değerlerle B-Tree Index Sıkıştırma

Beşinci denemede aslında yukarıda yaptığımız işlemi tekrar edeceğiz. Eğer sütun sayısı vermezseniz, NON-UNIQUE INDEX'ler için bütün sütunlar sıkıştırılır. UNIQUE index'lerde ise son sütun hariç bütün sütunlar için sıkıştırılma işlemi gerçekleştirilir. [1]

```
SET TIMING ON;
CREATE INDEX D_CCEBI.IDX_COMPRESS_COL_DEFAULT ON
D_CCEBI.DENEME( ID, VERI, SAYI )
COMPRESS
NOLOGGING
TABLESPACE TS_CCEBI
PARALLEL (DEGREE 4);
Index created.
Elapsed: 00:00:09.75

SELECT ROUND(bytes/1024/1024,2) INDEX_SIZE FROM DBA_SEGMENTS
WHERE OWNER='D_CCEBI' AND SEGMENT_NAME='IDX_COMPRESS_COL_DEFAULT';

INDEX_SIZE
----------
     4,5MB


-------------------------------------------------------------------------------------------------
| Id  | Operation        | Name                     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT |                          |  6268 |  6292K|    73   (0)| 00:00:01 |
|*  1 |  INDEX RANGE SCAN| IDX_COMPRESS_COL_DEFAULT |  6268 |  6292K|    73   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------
```

Görüldüğü üzere dördüncü deneme ile bir fark yok.

2.6. Bitmap Index

Seçiciliği az olan index'lerde bitmap index kullanmak bir başka yöntemdir. Bu sefer bir bitmap index yaratacağız.

```
SET TIMING ON;
CREATE BITMAP INDEX D_CCEBI.IDX_BITMAP ON D_CCEBI.DENEME( ID, VERI, SAYI )
NOLOGGING
```

[1] http://richardfoote.wordpress.com/2008/02/17/index-compression-part-i-low/

```
TABLESPACE TS_CCEBI
PARALLEL (DEGREE 4);
Index created.
Elapsed: 00:00:05.07

INDEX_SIZE
----------
    3,88MB


-----------------------------------------------------------------------------------------
| Id  | Operation                    | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |            |  6268 |  6292K|    63   (0)| 00:00:01 |
|   1 |  BITMAP CONVERSION TO ROWIDS|            |  6268 |  6292K|    63   (0)| 00:00:01 |
|*  2 |   BITMAP INDEX RANGE SCAN    | IDX_BITMAP |       |       |            |          |
-----------------------------------------------------------------------------------------
```

Yukarıda da bahsettiğim gibi seçiciliği az olan tablo sütunlarında bitmap index kullanmak yararlı bir opsiyon. Ancak Bitmap index için sıkıştırma şansımız yok. Gerçi düşük bir boyuta sahip olduğundan muhtemelen sıkıştırmaya benzer bir durum var.

2.7. Tablo Güncellemelerinde Bitmap ve Sıkıştırılmış B-tree Index

Testlerin sonunda bitmap index ile doğru sıkıştırılmış b-tree index'in düşük seçicilik olsa bile kapıştığını gördük. Peki yeni değer girme veya silme durumunda sonuç (index boyutları, kayıt girme/silme zamanları vs...) nasıl değişecek? Şimdi de bu sorunun cevabını arayalım:

Her iki index için de tablo drop edilip, önce 50bin kayıt girildi ve test başlangıç durumuna alındı. Ardından aşağıdaki ifade çalıştı:

```
set timing on
begin
    -- 30BIN YENI KAYIT GIRILIR
    FOR i IN 1..30000
    LOOP
         insert into D_CCEBI.DENEME values(mod(i,10),
            'Lorem ipsum dolor sit amet, consectetur adipiscing elit.',
            DBMS_RANDOM.VALUE( )*10 );
        if mod(i,10000)=0 then
            COMMIT;
        END IF;
    END LOOP;
    COMMIT;
    -- ID DEGERI 2'DEN UFAK OLANLAR SILINIR
    DELETE FROM D_CCEBI.DENEME WHERE ID < 2;
    COMMIT;
END;
```

Sıkıştırılmış b-tree index ile işlem 01 dk. sürdü. İşlem sonunda index boyutu 1.38MB'tan 3.38MB'a çıktı. Cost ise 24'ten 52'ye yükseldi.

Bitmap index ile işlem tekrar edildiğinde 01dk. 10sn. sürdü. Boyut 3.88'den 4.81'e çıktı. Cost 63'ten 90'a yükseldi.

3. SONUÇ

Yapılan çalışmalarda şunu gördük; seçiciliği az olan bir durum söz konusu ise normal bir b-tree index yaratmak sağlıklı değil. Bunun yerine bitmap kullanmak gerekiyor. Ancak sıkıştırılacak sütunları doğru seçerseniz, seçicilik düşük olsa bile B-tree index ile bitmap index'ten daha başarılı sonuçlar elde edebilirsiniz. Ayrıca B-tree index'i sıkıştırırken bütün sütunların sıkıştırılmasının sağlıklı sonuç vermediği görülüyor. Önemli olan bir block içinde yer alacak sütun eşlerinin hangisinin daha çok tekrar edeceğini kestirebilmek.

Diğer bir konu da tablonun işlem görmesi... Sıkıştırılan index eğer çok DML gören bir tablodaysa, index'i sıkıştırmak mantıklı değildir. Yerden kazanmak bu kadar önemliyse ve sıkıştırmakta kararlıysanız, zaman zaman index'i drop edip, tekrar yaratmayı tercih edebilirsiniz. Drop edip tekrar yaratma bitmap index'lerde sıkça kullanılan bir yöntemdir. Compress b-tree index'lerde de gerek fragmentasyonun önüne geçmek, gerekse alanı geri kazanmak için aynı metodu deneyebilirsiniz.

Güncellenen bir tabloda sıkıştırılmış b-tree index, ara ara yeniden yaratılmadığı (veya rebuild edilmediği) takdirde index'in boyut artışı kaçınılmaz olarak devam eder. Bu da cost artışına neden olacak; hatta bir süre sonra index yerine tabloya full gitme gibi durumlar gerçekleşecektir. Yine de sıkıştırılan b-tree index'in bitmap index'e göre bir avantajı vardır. Sık güncellenen bir tabloda sıkıştırılmış b-tree index'leri drop edip tekrar yaratmayı sıklıkla yapmasanız bile, bitmap index'e nazaran, güncellemelerde daha az işlem gücü gerekecektir.