ORACLE VERİTABANINDA TABLO ve INDEX SIKIŞTIRMA

İçerik

1.Giriş

Oracle veritabanında tablo ve index sıkıştırma özelliği sayesinde, disk alanından önemli ölçüde tasarruf etmek mümkündür. Oracle'a ait teknik dökümanlarda 12'de 1'e kadar ufalan tablolardan bahsedilmektedir[1]. Özellikle büyük bilgi ambarlarında 12 GB'lık bir tabloyu 1 GB olarak saklayabilmek büyük fayda getirecektir. Ayrıca sıkıştırma işleminin faydası bununla da sınırlı kalmaz; sorgulara ait cost'ların da önemli ölçüde düşmesini sağlayabilir.

2. Sıkıştırma Mantığı

Tekrar eden datanın defalarca yazması yerine, data'yı bir kere yazıp, diğer satırların bu data'yı işaret etmesi sağlanırsa, disk alanından önemli ölçüde kazanım sağlayabilirsiniz. Oracle'ın sıkıştırma mantığı da bu şekildedir. Sıkıştırma block bazında gerçekleşir. Bunun anlamı, uncompressed data'nın açılabilmesi için gereken bütün bilginin yine aynı block'ta tutulmasıdır. Aynı block içindeki aynı veriye sahip satır ve sütunlar, symbol table olarak adlandırılan datablock'un başına kaydedilir. Tekrar eden datanın geçtiği her satır, symbol table'daki alanı refere eder. Başındaki symbol table dışında, sıkıştırılmış bir block, normal bir datablock'tan pek farklı değildir. Veritabanında normalde yaptığınız her işlemi, bir fark sezmeden gerçekleştirebilirsiniz. Sıkıştırma ve sıkıştırılan veriyi açma, Oracle'ın sunucu tarafında lokal'de gerçekleşen bir işlemdir.

3. Sıkıştırma İşlemini Kullanabileceğimiz Durumlar

Bir tabloya, index'e sıkıştırma uygulayabilirsiniz. Ya da tablespace seviyesinde sıkıştırma istediğinizi de belirtebilirsiniz. Böylece ilgili tablespace'de yaratacağınız tablo ve index'ler varsayılan olarak sıkıştırılmış şekilde ortaya çıkacaktır.

Bir tabloyu sıkıştırılmış olarak yarattıktan sonra, ona girilecek yeni değerler, sıkıştırılmış olmaz. Örneğin daha sonra gerçekleştirilen insert ifadeleri sıkıştırılmaz, normal bir şekilde kaydedilir. Eğer bir bulk işlem söz konusu ise ve mutlaka compressed olmasını istiyorsanız, APPEND HINT kullanabilirsiniz. Ama kullanmadığınız sürece, compress bir tabloya girilen insert ifadesinin normal bir tabloya girilen bir insert ifadesinden hiçbir farkı yok. Bu nedenle yavaşlama olmayacaktır.

Compress özelliğini belirterek CTAS (**C**REATE **T**ABLE ... **AS**) şeklinde tablolar yaratırsanız, oluşturulacak tablo sıkıştırılmış şekilde yaratılacaktır. Daha önceden sıkıştırılmış bir tabloya

[1] **Table Compression in Oracle9*i* Relase2**

veri girilecekse, parallel insert (veya APPEND Hint'i ile insert) gerçekleştirildiğinde ve Direct Path SQL*Loader sayesinde veriyi sıkıştırılmış olarak aktarmanız mümkündür.

4. Sıkıştırılacak Tablolara Karar Vermek

Sıkıştırma özelliği ne kadar kulağa hoş gelse de; yanlış tablolarda kullanmak, hem fazla disk kullanımı hem de lüzumsuz yere CPU tüketimine yol açacaktır. Sıkıştırma için en ideal tablolar, log işlemleri için kullanılan, ağırlıklı olarak insert gören, update ve delete ile pek işi olmayan karaktere sahiptir.

Yazılım geliştirenler için hangi tablonun çok faal, hangisinin log amaçlı olduğunu bilmek kolay olsa da, bunu bilemeyebilirsiniz. Hangi tablonun sık update, hangisinin sık delete ya da hangisinin sık insert gördüğünü, **dba_tab_modifications** tablosundan çıkartmak mümkündür. Örneğin belirli bir schema altında Delete ve Update işleminin hiç yapılmadığı, insert konusunda kayda değer ilk beş tabloyu saptamak için aşağıdaki sorguyu kullanabilirsiniz:

```
SQL> SELECT * FROM SYS.DBA_TAB_MODIFICATIONS
     WHERE UPDATES=0 AND DELETES=0
     AND TABLE_OWNER='SCHEMA_ADI' AND ROWNUM <= 5
     ORDER BY INSERTS DESC;
```

Ya da tablo üzerindeki update ve delete'lerin toplam insert işleminin %20'sini geçmediği ilk 5 tabloyu saptamak için şöyle bir ifade kullanılabilir:

```
SQL> SELECT * FROM SYS.DBA_TAB_MODIFICATIONS
     WHERE INSERTS >= ( UPDATES + DELETES ) * 5
     AND UPDATES!=0 AND DELETES!=0
     AND TABLE_OWNER='SCHEMA_ADI' AND ROWNUM <= 5
     ORDER BY INSERTS DESC;
```

Bir tablonun %10'luk bir kısmı değiştiği takdirde, ***DBA_TAB_MODIFICATIONS*** tablosundaki ***TIMESTAMP*** sütunundaki değer değişecektir.

DBA_TAB_MODIFICATIONS'ta tablonuzun bulunabilmesi için, MONITORING seçeneğinin açık olması gerekir. 10G ile birlikte bu varsayılan (***default***) ayar hâlini almıştır. Daha eski bir sürüm kullanıyorsanız ve bazı tablolarınızda monitoring'in kapalı olduğundan şüpheniz varsa, aşağıdaki sorguyla kontrol edebilirsiniz:

```
SQL> SELECT * FROM DBA_TABLES
     WHERE OWNER='SCHEMA_ADI' AND
     MONITORING = 'NO';
```

Şayet tabloda monitoring kapalı hâldeyse, aşağıdaki gibi açabilirsiniz:

```
SQL> ALTER TABLE TABLO_ADI MONITORING;
```

Ekstra bir not; temporary table'larda monitoring kapalı gelir ve açamazsınız. Aksi de, pek mantıklı olmazdı.

5. Performans

Oracle tarafından yayınlanan "***Table Compression in Oracle9i Relase2***" dökümanına göre,

- Normal INSERT ifadelerinde, ölçülebilir bir fark ortaya çıkmamaktadır. Bunun nedeni, işlemin uncompressed şekilde gerçekleşmesidir. APPEND HINT vb. şekilde girilen ifadelerinse %50'ye varan oranda daha iyi olduğu durumlar çıkabilir. (Buna 7.bölümde tekrar değineceğiz.)
- DELETE işlemleri, sıkıştırılmış tablolarda %10 daha fazladır. Çünkü sıkıştırılmış hâlde tutulan veri, daha az yer kaplar; böyle log'lanacak daha az bilgi olur.
- Sıkıştırılmış tablolardaki UPDATE işlemlerinde, %10-20 arası bir performans kaybı yaşanır. Bunun nedeni, normal tablolarda kullanılan bazı kompleks update yöntemlerinin, compress tablolara henüz uyarlanmamasıdır. (Bu inceleme 9iR2 için yapılmıştı; 10G için bir dökümanda rastlamadım, 10g'de update performansı çok daha iyi olabilir.)

Yine Oracle tarafından yayımlanan bir başka dökümanda ("***Data Compression in Oracle***"), benzer sonuçlara yer verilmiş.

Sıkıştırılmış tablolarda, update ve delete konusunda bir iki noktanın daha altını çizmek gerekiyor. Bir data block'unun içindeki satırı güncellemeye kalktığınızda, Oracle, ilgili block'un başındaki symbol table'i girilen kayıta göre güncellemek zorundadır. Örneğin daha önce 4 adet mavi, 3 adet sarı kalemin olduğu bir kutudan 1 maviyi çıkartıp, yerine 2 adet yeşil kalem koydunuz. Oracle'ın burada yaptığı, bu kutunun içinde eskiden 4 mavi, 3 sarı varken; şimdi 3 mavi, 3 sarı ve 2 adet yeşil kalem olduğu bilgisini girmek gibi düşünülebilir.

Delete işleminin daha hızlı olduğunu yukarıda yazmıştık. Ancak delete işleminin bir sıkıntısı bulunuyor. İyi sıkıştırılmış bir tabloda, bir satırı delete edip, yeni bir insert yaparsanız, büyük bir ihtimalle aynı data block'u kullanmayacaktır. Çünkü satır diye sildiğimiz şey, gerçekten de bir satır değildi; symbol table'da bir değeri refere eden pointer gibi çok daha ufak boyuta sahip bir alandı. Dolayısıyla siz bu ufak alanı silip, yeni bir satır girmeye kalktığınızda, bu ufak alan yetersiz kalacak ve ister istemez tablonun işgal ettiği block sayısı artacaktır. Bir noktadan sonra, tablonun normalde kaplayacağı alandan çok daha büyük bir alanın işgali bile söz konusu olabilir. Bu nedenle, sık delete ve update işleminin yapıldığı tablolarda sıkıştırmak çok uygun bir seçenek değildir.

6. Var Olan Bir Tabloyu Sıkıştırmak

Loglama işlemi için kullanılan, hiç update ve delete görmeyen, tam sıkıştırmaya uygun bir tablo bulduğumuzu varsayalım. Bu durumda, tablonun sıkıştırarak yedeğini alarak işe başlayabiliriz:

```
/* HER IHTIMALE KARSI GECICI TABLO YARATILIR */
CREATE TABLE D_CCEBI.TABLO_YEDEGI_241108
TABLESPACE BACKUP_TABLESPACE
COMPRESS
NOLOGGING
AS SELECT * FROM SCHEMA.TABLO;
```

Dikkat ettiyseniz, tablonun yedeğini alırken, tablespace'i ***BACKUP_TABLESPACE*** olarak belirttim. Normalde kullandığınız tablespace yerine, Backup işlemleri için bir tablespace yaratıp, yedeklerinizi bunun altınıza almanız daha faydalı olur. Çalışma ortamınızı genişletmemiş olursunuz. İkinci dikkat edilecek konu ise, ***NOLOGGING*** mode belirtmemdi. Eğer **nologging mode** belirtirseniz, tablo kopyası daha kısa sürede oluşacaktır. Fakat... Bir çökme durumunda bu tablonun içindeki verileri, standby'da ya da arşiv dosyalarında bulamazsınız. Bu riski göze alarak işleme devam edersek, orjinal tabloyu aşağıdaki gibi sıkıştırabiliriz:

```
/* TABLO SIKISTIRILIR */
SQL> ALTER TABLE SCHEMA.TABLO MOVE COMPRESS;
```

Bu işlemi de nologging ile yapmamız mümkündür:

```
/* TABLO NOLOGGING MODE'DA SIKISTIRILIR */
SQL> ALTER TABLE SCHEMA.TABLO MOVE COMPRESS NOLOGGING;
```

İşleme başlamadan önce yedek alırken, NOLOGGING mode kullanmak uygundur. Fakat gerçek bir ortamda sıkıştırma yapacağınız tabloda ***NOLOGGING*** mode kullanmak pek akılcıl bir yöntem olmayabilir.

Tabloyu sıkıştırarak ne kadar yer kazandığınızı görmek için aşağıdaki sorguyu işlem başında ve sonunda çalıştırabilirsiniz:

```
SQL>  SELECT SEGMENT_NAME,
             TABLESPACE_NAME,
             ROUND(BYTES / 1024 / 1024, 2) SIZE_MB
       FROM DBA_SEGMENTS
      WHERE OWNER='SCHEMA' AND SEGMENT_NAME = 'TABLO'
      AND SEGMENT_TYPE = 'TABLE'
      ORDER BY 3 DESC;
```

Tablo üzerinde index varsa, tablonun sıkıştırılması index'leri kullanılamaz hâle getirebilir. İşlem öncesi hangi index'lerin etkileneceğine bakmak yararlı olacaktır.

```
SQL> SELECT * FROM DBA_INDEXES
     WHERE OWNER='SCHEMA' AND TABLE_NAME='TABLO';
```

İşlem sonrası kullanılamaz index'leri aşağıdaki sorguyla kontrol edebilirsiniz:

```
SQL> SELECT * FROM DBA_INDEXES
     WHERE STATUS='UNUSABLE';
```

Kullanılmaz index'ler söz konusu ise aşağıdaki gibi rebuild etmeniz mümkündür:

```
SQL> ALTER INDEX SCHEMA.INDEX_ADI
     REBUILD
     COMPRESS
     NOPARALLEL
     TABLESPACE DENEME_TABLESPACE
     STORAGE ( INITIAL 4M );
```

Index'i rebuild ederken sıkıştırmanız şart değil; sadece bir opsiyon. Index'i rebuild etmeniz de şart değil; o da bir başka seçenek. Dilerseniz drop eder ve tekrar yaratabilirsiniz.

Compress sonrası tabloyla ilgili (mesela %20 örneklemeyle) istatistik toplamanız yararlı olacaktır:

```
SQL> ANALYZE TABLE SCHEMA.TABLO
     ESTIMATE STATISTICS SAMPLE 20 PERCENT;
```

Bütün bu işlemler sonucunda kullandığınız sorguların execution plan değişebilir. Sorguları işlem sonunda kontrol etmek bu yüzden önemli olacaktır.

Sıkıştırılmış tablonun içeriğinin tamamen sıkıştırılmış olması gerekmez. Tablo içindeki verinin bir kısmı sıkıştırılmış, bir kısmı normal bir biçimde tutulabilir. Eğer normal yöntemlerle insert ifadeleri çalıştırılıyorsa; sıkıştırılmış tabloyu tekrar sıkıştırabilirsiniz. Aynı adımları takip etmeniz gerekmektedir.

7. Sıkıştırılmış ve Normal Tablo Farkının Uygulamalı Gösterilmesi

Şimdi kapsamlı bir örnek yapalım. Önce normal bir tablo oluşturup, bunun boyutunu alalım:

```
SQL> CREATE TABLE D_CCEBI.DENEME
     NOCOMPRESS
     AS
     SELECT * FROM DBA_OBJECTS;
     Table created.
```

NOCOMPRESS şeklinde belirtmemiz gerekmiyordu. Tablo yaratılırken, varsayılan değer tablonun sıkıştırılmamasıdır. Ancak yaptığımız iş daha açık gözüksün diye böyle yazmayı tercih ettim. Sıkıştırmadan yarattığımız tablonun boyutu, 6 MB oldu. Şimdi aynı tabloyu sıkıştırarak yaratalım:

```
SQL> CREATE TABLE D_CCEBI.DENEME
     COMPRESS
     AS
     SELECT * FROM DBA_OBJECTS;
     Table created.
```

İfade hemen hemen aynı olmasına rağmen, tablonun boyutu 2 MB'a (ilk tablonun %33'üne) indi. Şimdi örneğimizi biraz daha geliştirelim. Bu sefer compress olarak bir tablo yaratalım ve sonradan insert ifadeleri çalıştıralım. İlk yöntemde insert ifadeleri bildiğimiz sıradan şekilde olsun, ikincisinde APPEND HINT'ten yararlanalım.

```
/* BOS SEKILDE TABLOYU COMPRESS OLARAK OLUSTURUYORUZ */
SQL> CREATE TABLE D_CCEBI.DENEME
     COMPRESS
     AS
     SELECT * FROM DBA_OBJECTS
     WHERE 1 = 2;
```

Önce tabloya normal bir şekilde insert işlemi gerçekleştiriyoruz:

```
SET TIMING ON
BEGIN
    FOR i IN 1..50
    LOOP
        INSERT INTO D_CCEBI.DENEME
        SELECT * FROM DBA_OBJECTS;
        COMMIT;
    END LOOP;
END;

PL/SQL procedure successfully completed.
Elapsed: 00:00:54.46
```

Şimdi de tabloya APPEND hint ile değer girelim:

```
SET TIMING ON
BEGIN
    FOR i IN 1..50
    LOOP
        INSERT  /*+ APPEND */ INTO D_CCEBI.DENEME
        SELECT * FROM DBA_OBJECTS;
        COMMIT;
    END LOOP;
END;
PL/SQL procedure successfully completed.
Elapsed: 00:00:47.12
```

İşlem 47 saniye içinde tamamlandı ve tablo 96 MB yer işgal etti. Normal insert'e göre zaman daha kısa sürdü. Boyut olarak ise 160MB kârdayız. Boyut konusunda olumlu bir iyileşme bekliyorduk ancak insert işlemini daha kısa sürede tamamlamasını incelenmemiz yerinde olacak.

APPEND HINT ile yapılan INSERT'ler sıkıştırma işlemine tabiidir. Veri önce sıkıştırılır, ardından diske yazılırlar. Bu da elbette bir maliyettir.[2] Ancak diske yazmak vb. I/O işlemleri daha büyük maliyetler getirmektedir. Örneğin normal bir insert işlemi, 100 birim boyutunda olsun ve her birimi diske yazmak 1 iş gücü gerektirsin; bu durumda insert ifadesi 100 birim maliyete sahiptir. Fakat sıkıştırma işlemi 100 birimi, 10 birime düşürüyorsa, bunun diske yazım maliyeti 10 birimdir. Sıkıştırma için de 20 birim maliyet belirlesek, bütün işlem 100 birim yerine sadece 30 birimde tamamlanır. Bu düşünsel benzetmeyi bir kenara bırakıp, gerçeklere dönersek; diske erişim maliyetli bir iş ve yazılacak veriyi küçültmek maliyeti (çalışma süresini, tüketilen kaynakları) azaltacaktır. Özellikle toplu işlerde bunun daha fazla yararı görülebilir.

[2] **Table Compression in Oracle9*i* Relase2**