ORACLE
BELLEK YÖNETİMİYLE İLGİLİ NOTLAR

İçindekiler

Bir tabloda sürekli olarak belirli satırları sorguluyor ya da değiştiriyorsanız, her seferinde diske gitmeniz mantıklı olmaz. Çünkü I/O maliyetli bir işlemdir. Bu yüzden bellek performans için can simididir. Sıkça okuduğunuz ya da sıkça değiştirdiğiniz şeyleri her seferinde diske yazmak ya da diskten okumak yerine bunu bellekte kotarabilirseniz, performansı arttırabilirsiniz.

Bellekte çalışmak performans konusunda artış sağladığından Oracle da böyle yapar; diskte çalışmak yerine mümkün olduğunca belleği kullanır. İşte bu yüzden bir sorguyu ikinci defa çalıştırdığınızda, sonuçlar daha hızlı gelir. Çünkü sorguda kullanılan block'lar hâlâ 'sıcaktır' ve diske hiç gitmeden, direkt bellekten sonuç döner. Bellekte yer kalmadığında, daha sıcak (kullanım sıklığı daha fazla) olan block'lar, soğuk (kullanım sıklığı daha az) block'ların yerini alır. Oracle kapalı bir yazılım olduğundan arka plânda ne olduğunu elbette bilemiyoruz. Ama zekice yazılmış bir ***Least Recently Used – LRU*** algoritmasıyla bu işin yapıldığını tahmin edebiliriz.

İşlemlerin bellekte gerçekleştirilmesi güzel bir olay. Ama bazı durumlarda (performans problemleriyle karşı karşıya kaldığımızda ya da testlerin daha tutarlı olması gerektiğinde) Oracle'a bazı komutlarla müdahele etmemiz gerekir. Bu yazıda bizim için önemli bazı temel komutlardan ve Oracle'a etkilerinden bahsedeceğiz. Önce üzerinde duracağımız komutları verelim, ardından bu komutların bellekte yaratacağı etkileri inceleyelim.

a. ALTER SYSTEM FLUSH BUFFER_CACHE;
b. ALTER SYSTEM FLUSH SHARED_POOL;
c. ALTER SYSTEM CHECKPOINT;
d. i. ALTER SYSTEM SWITCH LOGFILE;
   ii. ALTER SYSTEM ARCHIVE LOG CURRENT;

1. BUFFER CACHE

Oracle'da birçok şey sadece bellekte tamamlanıyor. Örneğin bir DML işlemi (insert, update, delete) yaptığınızda gidip veritabanı dosyalarına yazması gerekmez. Ya da bir sorgu çalıştırdığınızda, sorgu biter bitmez, veri saklayan block'ların bellekten atılması gerekmez. Block'lar sorgulara göre bellekte tutulabilir. Hatta çalıştırdığınız

sorgular, diske hiç erişmeden sonuç dönüyor olabilir. Bununla ilgili bir örnek üzerinden gidelim. Önce aşağıdaki gibi tabloyu yaratıyoruz:

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

Table created.
```

Yarattığımız tabloda toplamda 72.526 kayıt var ve tablonun bulunduğu tablespace'te her block 8K'dan oluşuyor. DBA_SEGMENTS'ten baktığımda tablonun 8.388.608 byte (8MB) kapladığını ve 1024 block'tan oluştuğunu görüyorum. Tablo istatistiklerini topladığımdaysa, 1005 block görünüyor. (Block boyutunu 1K'ya kadar indirip, PCTFREE değerini düşürebilseydik; bu ikisi arasındaki fark daha da ufak olabilirdi.)

Sırada basit bir sorgu yazmak var:

```
SQL> SELECT COUNT(*) FROM D_CCEBI.DENEME WHERE OBJECT_ID > 1000;

  COUNT(*)
----------
     71479
```

Şimdi de sorgunun ne kadar disk okuması yaptığına bakalım.

```
SELECT a.SQL_ID, A.DISK_READS, A.EXECUTIONS, A.BUFFER_GETS
FROM V$SQL a WHERE a.SQL_TEXT LIKE '%SELECT COUNT(*) FROM
D_CCEBI.DENEME WHERE OBJECT_ID > 1000%'

SQL_ID          DISK_READS EXECUTIONS BUFFER_GETS
--------------- ---------- ---------- -----------
9szcxcccznnfnq         1011          1        1134
```

Buradaki ***DISK_READS***, okunan block sayısı; ***EXECUTIONS*** ifadenin kaç kere çalıştığı; ***BUFFER_GETS*** ise SQL ifadesi nedeniyle bellekten ne kadar block okunduğunu gösteriyor. Bunları daha iyi görebilmek için aynı sorguyu defalarca çalıştırıyorum:

```
SQL_ID          DISK_READS EXECUTIONS BUFFER_GETS
--------------- ---------- ---------- -----------
9szcxcccznnfnq         1011         33       32614
```

Sorguyu defalarca çalıştırdıktan sonra, EXECUTIONS sayısı arttığı hâlde DISK_READS artmıyor, fakat BUFFER_GETS ciddi bir artış gösteriyor. Yani SQL'in diske gitmeden sürekli bellekten sonuç döndüğünü görebiliyoruz. Bu güzel bir özellik ama eğer bir SQL'i test ediyor olsaydım, sonuçları sürekli bellekten okumak beni yanıltırdı. Bu gibi durumlar için bellekteki block'ları boşaltmak daha doğru sonuçlar elde etmemi sağlar. Deneyelim, sorguyu tekrar çalıştıralım:

```
SQL> ALTER SYSTEM FLUSH BUFFER_CACHE;

SQL_ID          DISK_READS EXECUTIONS BUFFER_GETS
--------------- ---------- ---------- -----------
9szcxcccznnfnq         1975         34       33600
```

Özetle ***FLUSH BUFFER_CAHCE*** yaparsanız;

- Bellekte tutulan block'lar atılır.
- Dirty Buffers (değişime uğramış ama diske yazılmamış, bellekte tutulan block'lar) diske yazılır
- SQL Plan bozulmaz
- SQL ile ilgili istatistikler silinmez

Bu dediklerimi **V$SQL_PLAN** view'ini sorgulayarak siz de görebilirsiniz:

```
SELECT * FROM V$SQL_PLAN
WHERE SQL_ID='9szcxcccznnfnq' AND ID=0;
```

(* ***SQL_ID*** *kullanacağınız sorguya göre değişiklik gösterecektir*.)

2. SHARED POOL

İlk kez çalışan SQL ifadeleri hard parse yapılır. SQL ifadesi hash'lenerek ona özel bir kimlik (SQL_ID) verilir; ihtimaller hesaplanarak en uygun (cost'u en düşük) plan belirlenir. Bu masraflı bir iştir; CPU'ya maliyeti vardır. Bu yüzden plan tekrar tekrar hesaplanmaz. Aynı ifade söz konusu ise aynı plan üzerinden gidilir. Bu güzel bir özellik ama arada ciddi sorunlar ortaya çıkarıyor. Örneğin ilk hesaplanan plan, sonradan gelen ifadeler içinde geçerli olur.

Şöyle düşünelim; bind değişken kullanan bir sorgu var. Ve ilk defa bu sorguyu kullanacağınızda çok geniş bir aralık verdiniz. Çok geniş aralık verildiğinde, önce

index'lere gidip, sonra tablo block'larına erişmek akıllıca olmayacaktır. Bu yüzden tabloya full gidecektir. Sizden sonra aynı ifadeyi kullanan fakat bind değişkenlerde çok kısa bir aralık veren ikinci bir sorgu gelebilir. Ve bu aralık ne kadar kısa olursa olsun, tabloya full yine full gidilecektir. Index'leri kullandığında çok çok daha hızlı gidecek olmasına rağmen, ikinci bir defa plan hesaplanmayacağından bunun farkına varamaz. Bundan sonraki bütün ifadeler de böyle çalışacaktır. Bu gibi durumlardan sakınmak için shared pool'u boşaltmak bir çözümdür:

```
SQL> ALTER SYSTEM FLUSH SHARED_POOL;
```

Yukarıda anlattıklarım zaten bilinen şeyler. Fakat göstermek istediğim bir durum var. Aşağıdaki sorguyu çalıştırırsanız artık sonuç boş gelecektir –ki bu beklediğimiz bir şey. Çünkü SQL ID ve buna bağlı her şey gidiyor.

```
SELECT a.SQL_ID, A.DISK_READS, A.EXECUTIONS, A.BUFFER_GETS FROM V$SQL a
WHERE a.SQL_TEXT LIKE
      '%SELECT COUNT(*) FROM D_CCEBI.DENEME WHERE OBJECT_ID > 1000%'
no rows selected.
```

Şimdi ilk sorgumuzu tekrar çalıştırıp, disk okumalarına bakalım:

```
SQL_ID          DISK_READS EXECUTIONS BUFFER_GETS
--------------- ---------- ---------- -----------
9szcxcccznnfnq            0          1        1016
```

İlk defa çalıştırıldığı ve sql plan ilk defa hazırlandığı hâlde, hiç disk okuması yapılmıyor.

Özetle ***FLUSH SHARED_POOL*** yaparsanız;

- Bütün SQL planları ve bellekte tutulan SQL bilgilerini atarsınız.
- Ancak bu buffer cache'te tutulan data block'larını etkilemez. Daha önce eriştiğiniz block'lar hâlâ bellektedir.

NOT: Shared Pool'u boşaltmak gerekli olmadıkça yapılmamalı. Shared Pool'u boşalttıktan sonra her sql için yeniden plan hesaplanacağından performans sorunu doğabilir.

3. CHECKPOINT

Buffer cache ile shared pool arasındaki bağı ortaya çıkartınca, checkpoint konusuna değinmeden olmaz. Checkpoint dirty buffer'ları diske yazmaya yarayan bir işlemdir. Yazının başında belirtmiştim, değişiklik olan data hemen diske yazılmaz; belirli

aralıklarda diske (veritabanı dosyalarına) çıkar. CHECKPOINT komutundan sonra, sql ifadesini birkaç kere çalıştırıp, yeni bir test yaptım:

```
SQL> ALTER SYSTEM CHECKPOINT;

SQL_ID          DISK_READS EXECUTIONS BUFFER_GETS
--------------- ---------- ---------- -----------
9szcxcccznnfnq            0          5        4960
```

Gördüğünüz gibi dirty buffer'ların diske yazılmasını sağlayan checkpoint, hot block'ların bellekten atılmasına neden olmuyor. Peki başka bir şey deneyelim; okuma yaptığımız block'larda değişiklik yapalım:

```
delete from d_ccebi.deneme
where object_id > 1000 and object_id < 50000;
commit;
22925 rows deleted.
Commit complete.

SELECT COUNT(*) FROM D_CCEBI.DENEME WHERE OBJECT_ID > 1000;

  COUNT(*)
----------
     68191
```

Sorguyu birkaç kere çalıştırıp, disk okuma durumunu tekrar kontrol ediyoruz:

```
SQL_ID          DISK_READS EXECUTIONS BUFFER_GETS
--------------- ---------- ---------- -----------
9szcxcccznnfnq            0         10        9890
```

Kayıt silmeme rağmen, hiç disk okuması yapılmadı. Şimdi checkpoint yapalım, sorguyu tekrar çalıştırıp, duruma tekrar bakalım:

```
SQL_ID          DISK_READS EXECUTIONS BUFFER_GETS
--------------- ---------- ---------- -----------
9szcxcccznnfnq            0         11        9890
```

Bu noktada beklediğim sonuç gelmedi. Dirty buffer'ı diske yazıyorsa,neden disk okuması yapmıyor diye düşündüm. Ancak aslında sonuçlar doğru. Çünkü dirty buffer'i diske yazması, yazılan block'ların bellekten atılmasına neden olmuyor.

Özetle checkpoint yaparsanız;

- Dirty buffer olarak tabir edilen değişmiş block'lar diske yazılır.

- SQL Plan, bellekteki okuma amaçlı block'lar bundan etkilenmez.

4. REDOLOG DOSYALARININ DEĞİŞİMİ

ALTER SYSTEM SWITCH LOGFILE veya ALTER SYSTEM ARCHIVE LOG CURRENT komutları checkpoint yapar ve online redo log dosyasını değiştirir. Bu iki komut temelde aynı işi yapar (checkpoint atılması, redolog switch, arşive çıkılması vs...); ama aralarında önemli bir fark vardır. ALTER SYSTEM SWITCH LOGFILE redolog dosyasının değişmesini beklemez; işlemlerin bitmesine bakmadan size hemen döner. ALTER SYSTEM ARCHIVE LOG CURRENT ise daha güvenlidir, işlem gerçek anlamda tamamlanmadan size yanıt dönmeyecektir. Yalnız LOG CURRENT ifadesinde NOLOGGING mode'da kullanamazsınız.

5. ORACLE BELLEK YÖNETİMİ VE ÇÖKMEYE KARŞI GÜVENLİĞİ

Aslında Oracle'ın bellek yönetimi arkasındaki felsefe basit. Yapabildiğin bütün işleri bellekte yap; yapamadıkların için diski kullan. Misal çok uzun süren işlemlerde, commit yapmasanız bile LGWR redo buffer içeriğini peyderpey redo log dosyalarına aktarır. Block'lardaki değişikliği ise datafile'lara yazar. Bir şeyin datafile'a ya da redolog dosyasına yazılması, onun kalıcı olduğu anlamına gelmez. Siz ne zaman commit yaparsanız, değişiklik o zaman kalıcı olur. Ama kalıcı olan değişikliklerin de datafile'larda bulunması gerekmez. Önemli olan redolog'un commit ifadesini görmesidir. Gerektiğinde (mesela veritabanının aniden kapanması), redo log'tan yapılan işlemler okunacak ve datafile'a kaydedilmemiş dosyalar kaydedilecektir. Bu yüzden commit görmeyen büyük ifadeler çalıştırmaktan sakınmak lâzım. Bir çökme durumunda bütün iş tekrarlanacak, sonunda commit görmediğinden, yapılanlar geçersiz sayılacaktır.