DİNAMİK ve STATİK SQL KULLANMANIN PERFORMANSA ETKİSİ

İçindekiler

1. SQL Yazımında Dikkat Edilecekler

SQL ifadelerini çeşitli nedenlerle dinamik yazmamız gerekebilir. Ancak dinamik yazılacak SQL ifadelerinin işlenmesi (parse) ve buna göre bir çalışma planı (execution plan) hazırlanması önemli maliyettir. Örneğin dinamik bir ifade ile statik yazılan bir ifadenin performans farkı aşağıda net olarak görülebilir:

```
-- TABLO YARATIYORUZ
CREATE TABLE D_CCEBI.DENEME ( ID NUMBER );

-- HARD PARSE
set timing on
set serveroutput on
DECLARE
    i   PLS_INTEGER;
BEGIN
    FOR i IN 1..500000 LOOP
       EXECUTE IMMEDIATE 'INSERT INTO D_CCEBI.DENEME VALUES( '|| i || ')';
    END LOOP;
    COMMIT;
END;
PL/SQL procedure successfully completed.
Elapsed: 00:06:27.51
```

Şimdi de bu işlemi statik olacak şekilde değiştirelim.

```
-- TABLOYU SİLİYORUZ
truncate table d_ccebi.deneme;

-- PROSEDÜR YARATALIM
CREATE OR REPLACE PROCEDURE D_CCEBI.DENEME_INSERT ( ID NUMBER )
AS
BEGIN
    INSERT INTO d_ccebi.deneme values( id );
END;

-- PROSEDÜR ÜZERİNDEN KAYIT GİRİYORUZ
set timing on
set serveroutput on
DECLARE
    i   PLS_INTEGER;
BEGIN
    FOR i IN 1..500000 LOOP
       D_CCEBI.DENEME_INSERT( i );
    END LOOP;
    COMMIT;
END;
PL/SQL procedure successfully completed.
Elapsed: 00:00:32.07
```

Görüldüğü gibi tekrar tekrar ifadenin işlenmesine engel olursak çok ciddi bir performans artışı sağlayabiliyoruz. Söz konusu OLTP bir ortamsa, bind değişken kullanımı –yukarıdaki örnekte de görüldüğü gibi çok yararlı olabilir. Ancak her zaman

statik yazmak ya da bind değişken kullanmak çok güzel sonuçlar vermeyecektir. Bazı durumlar da dinamik sql ifadelerine ihtiyaç duyuyoruz. Örneğin bir datawarehouse uygulamasında, saatlerce süren sorgular düşünüldüğünde verilen değerlere göre execution plan'ın tekrar tekrar hesaplanması çok daha verimli olabilir.

Java vb. modern programlama dillerinde SQL ifadelerini program içinden dinamik şekilde yazabilirsiniz. Bunu PL/SQL ile yapmak da mümkün. Bu yazıda dinamik ve statik sql örnekleriyle, execution plan arasındaki bağı açıklamaya çalışacağım.

2. TAMAMEN DİNAMİK SQL ÖRNEĞİ

Basit bir test yapıp, DBA_OBJECTS tablosundan nesne numarasına göre bazı bilgileri sorgulayan ifadeleri oluşturacağız. Ancak bunu yapmadan önce sonuçları daha iyi analiz etmek için shared pool'u flush etmemiz yararlı olur:

```
-- SHARED POOL FLUSH EDILIR
SQL> ALTER SYSTEM FLUSH SHARED_POOL;

-- IFADEYI OLUSTURUYORUZ
CREATE OR REPLACE PROCEDURE query_obj( obj_id varchar ) IS
    TYPE cv_type IS REF CURSOR;
    cv            cv_type;
    obj_name      varchar2(30);
    query_str     varchar2(1000);
    val           varchar2(32000);
    ob_id         number;
BEGIN
    query_str := 'SELECT OBJECT_NAME, OBJECT_ID FROM DBA_OBJECTS WHERE
object_id ='||OBJ_ID;
    OPEN cv FOR query_str;
    LOOP
        FETCH cv INTO val, ob_id;
        EXIT WHEN cv%NOTFOUND;
          null;
    END LOOP;
    CLOSE cv;
END;

SET TIMING ON
set serveroutput on
begin
    FOR i in 1..10000 LOOP
        query_obj( TO_CHAR( i ) );
    END LOOP;
end;
PL/SQL procedure successfully completed.
Elapsed: 00:01:00.75
```

Oluşturulan SQL ifadelerini ve bunlara ait execution plan'ları görmek için aşağıdaki sorguları kullanıyoruz:

```
select count(*) from v$sqlarea where sql_text like 'SELECT OBJECT_NAME,
OBJECT_ID FROM DBA_OBJECTS WHERE%'

select count(*) from v$sql_plan where sql_id in
( select sql_id from v$sqlarea where sql_text like ' SELECT OBJECT_NAME,
OBJECT_ID FROM DBA_OBJECTS WHERE%' )and id = 0;
```

Çalıştırdığımız ifade sonucunda 10bin adet ayrı execution plan oluşuyor ve işlemin yavaş olduğunu görüyoruz. Ancak dikkat etmemiz gereken bir durum daha var: 4000.adımdan sonra eski planlar hızla eziliyor. (Kullandığınız bellek boyutuna ve ortamın yoğunluğuna göre hiç ezilmemesi de mümkün...)

3. DİNAMİK SQL İÇİN CURSOR_SHARING'I FORCE ETMEK

Bind değişken kullanmanın çok iyi sonuçlar yaratacağı bazı durumlarda kodun içine müdahele şansımız olmayabilir. Bu gibi durumlarda **cursor_sharing** parametresi kullanarak aynı SQL ifadelerinin bind değişken kullanmasını '*zorlayabiliriz*'. Normalde cursor_sharing exact değeriyle gelir; ancak bunu force ederek, aynı yazılmış SQL'lerde bind değişken atamış gibi davranmamız mümkündür. Şimdi aynı örneği **cursor_sharing** parametresini session bazında force ederek tekrar ediyoruz. (Doğru analiz için her test öncesinde flush işlemini mutlaka yapmamız gerektiğini unutmamak gerekiyor.)

```
CREATE OR REPLACE PROCEDURE query_obj( obj_id varchar) IS
    TYPE          cv_type IS REF CURSOR;
    cv                  cv_type;
    obj_name            varchar2(30);
    query_str           varchar2(1000);
    val                 varchar2(32000);
    ob_id               number;
BEGIN
    query_str := 'SELECT OBJECT_NAME, OBJECT_ID FROM DBA_OBJECTS WHERE
object_id ='||OBJ_ID;
    OPEN cv FOR query_str;
    LOOP
        FETCH cv INTO val, ob_id;
        EXIT WHEN cv%NOTFOUND;
          null;
    END LOOP;
    CLOSE cv;
END;
```

```
-- SESSION BAZINDA CURSOR_SHARING YAPIYORUZ
ALTER SESSION SET CURSOR_SHARING = FORCE;
SET TIMING ON
set serveroutput on
begin
    FOR i in 1..10000 LOOP
        query_obj( TO_CHAR( i ) );
    END LOOP;
end;
Session altered.
PL/SQL procedure successfully completed.
Elapsed: 00:00:02.32
```

Sadece tek bir execution plan oluştu, bu yüzden bariz bir hızlanma var.

```
SELECT OBJECT_NAME, OBJECT_ID FROM DBA_OBJECTS WHERE object_id =:"SYS_B_0"
```

4. DİNAMİK PL/SQL BLOĞUNDA BIND DEĞİŞKENLE ÇALIŞMAK

Dinamik SQL ifadeleri içinde bind değişken kullanmak mümkündür.

```
CREATE OR REPLACE PROCEDURE query_obj( obj_id varchar ) IS
    TYPE          cv_type IS REF CURSOR;
    cv            cv_type;
    obj_name      varchar2(30);
    query_str     varchar2(1000);
    val           varchar2(32000);
    ob_id         number;
BEGIN
    query_str := 'SELECT OBJECT_NAME, OBJECT_ID FROM DBA_OBJECTS WHERE
object_id = :OBJ_ID';
    OPEN cv FOR query_str USING obj_id;
    LOOP
        FETCH cv INTO val, ob_id;
        EXIT WHEN cv%NOTFOUND;
          null;
 END LOOP;
    CLOSE cv;
END;

SET TIMING ON
set serveroutput on
begin
    FOR i in 1..10000 LOOP
        query_obj( TO_CHAR( i )  );
    END LOOP;
end;
PL/SQL procedure successfully completed.
Elapsed: 00:00:02.15
```

Yukarıdaki PL/SQL bloğu bind değişken kullanan dinamik bir sql örneğidir. İfade dinamik olduğu hâlde bind değişken sayesinde tek bir execution plan oluşur ve performans çok ciddi oranda artar.

```
SELECT OBJECT_NAME, OBJECT_ID FROM DBA_OBJECTS WHERE object_id =
:OBJ_ID
```

Elbette burada önemli bir soru aklımıza takılabilir: "*Madem bind değişken kullanıyoruz, neden ifadeyi statik yazmıyoruz?*"

Yukarıdaki ifade çok basit bir örnek, ne gelirse gelsin, her zaman aynı ifadeyi çalıştırıyor. Hâlbuki gelen ifadeye göre SQL hint veriyor olsaydık, durum daha net anlaşılırdı. Örneğin gönderilen değer 0 ile 1000 arasında gelirse INDEX hint'i verebileceğimiz, 1000'den büyükse FULL gitmesini isteyeceğimiz bir şey yapmayı deneyebiliriz:

```
CREATE OR REPLACE PROCEDURE query_obj( obj_id varchar ) IS
    TYPE                cv_type IS REF CURSOR;
    cv                  cv_type;
    obj_name            varchar2(30);
    query_str           varchar2(1000);
    val                 varchar2(32000);
    ob_id               number;
    hint_statement      varchar2(200);
BEGIN

    IF obj_id >= 0 AND obj_id <= 1000 THEN
        hint_statement:='/*+INDEX( DBA_OBJECTS )*/';
    ELSE
        hint_statement:='/*+FULL( DBA_OBJECTS )*/';
    END IF;
    query_str := 'SELECT '|| hint_statement ||' OBJECT_NAME, OBJECT_ID FROM
DBA_OBJECTS WHERE object_id = :OBJ_ID';
    OPEN cv FOR query_str USING obj_id;
    LOOP
        FETCH cv INTO val, ob_id;
        EXIT WHEN cv%NOTFOUND;
          null;
 END LOOP;
    CLOSE cv;
END;


SET TIMING ON
set serveroutput on
begin
    FOR i in 1..10000 LOOP
        query_obj( TO_CHAR( i )  );
    END LOOP;
end;
PL/SQL procedure successfully completed.
Elapsed: 00:00:02.81
```

Yaratılan planları kontrol ettiğimizde iki tane SQL ifadesi yakalıyoruz:

```
SELECT /*+INDEX( DBA_OBJECTS )*/ OBJECT_NAME, OBJECT_ID FROM
DBA_OBJECTS WHERE object_id = :OBJ_ID

SELECT /*+FULL( DBA_OBJECTS )*/ OBJECT_NAME, OBJECT_ID FROM
DBA_OBJECTS WHERE object_id = :OBJ_ID
```

Verdiğimiz örnekler DBA_OBJECTS view'i ile ilgili olduğundan cost ile ilgili sonuçlar farklı olmuyor. Ancak verilecek hint'e göre çok farklı execution plan'lar oluşturmak mümkün. Ve gönderilen değere göre seçicilik (selectivity) değişeceğinden, bunun avantaj sağlayacağı birçok durum yaratılabilir.

5. TAMAMEN STATİK ÇALIŞMAK

OLTP olmak üzere tasarlanan veritabanlarında en çok kullanacağımız/kullandığımız durumdur.

```
CREATE OR REPLACE PROCEDURE query_obj( v_object_id varchar ) IS
    obj_name varchar2(30);
    obj_id   number;
    CURSOR C IS
        SELECT object_name,object_id FROM DBA_OBJECTS WHERE
            object_id=v_object_id;
BEGIN
    OPEN c;
    LOOP
        FETCH c INTO obj_name, obj_id;
        EXIT WHEN c%NOTFOUND;
         null;
    END LOOP;
    CLOSE c;
END;
/

SET TIMING ON
set serveroutput on
begin
    FOR i in 1..10000 LOOP
        query_obj( TO_CHAR( i )  );
    END LOOP;
end;
PL/SQL procedure successfully completed.
Elapsed: 00:00:01.21
```

Oluşan tek execution plan aşağıdaki içindir:

```
SELECT OBJECT_NAME,OBJECT_ID FROM DBA_OBJECTS WHERE OBJECT_ID= :B1
```

Bütün uygulamalar arasında en hızlı çalışan tamamen statik yazılmış kod oldu.

6. STATİK VEYA DİNAMİK KOD YAZIMINA KARAR VERMEK

Denemelerden de görüldüğü gibi BIND değişkenlerle çalışmak çok daha hızlı sonuç elde etmemizi sağlar; çünkü execution plan tekrar hesaplanmaz. Dinamik yazılmış sql ifadelerine gelirsek, bunlar için de bind değişken kullanmamız mümkündür. Ya da yazılımcı bunu yapmazsa, session bazında cursor_sharing'i force edip, bind değişken kullanımını zorlayabiliriz. Yine çok daha performanslı çalıştığı görülebilir.

Statik ya da dinamik kod yazarken çok önemli bir ayrıma gitmemiz gerekiyor: Bind değişken kullanmak her zaman güzel sonuçlar vermeyecektir. Özellikle büyük raporların döndüğü bir sorgu ise bind değişkenin yarardan çok zararı olabilir. Örneğin bind değişken kullanan bir sorguda, sisteme bağlanan ilk kullanıcı 1 yıllık bir sorgu çalıştırdı ve ilgili tabloya full gitti diye düşünelim. Bu durumda bundan sonra siz tek gün dahi çalıştırsanız, tabloya full gitmeye devam edersiniz. Hâlbuki dinamik yazılan SQL ifadesi olsa, her seferinde plan tekrar gözden geçirileceğinden çalışma planı daha doğru olacaktır.

Statik ya da dinamik kod yazma ayrımı yaparken, çalıştırılacak ifadenin ne kadar veri çekeceği, kimler tarafından kullanılacağı, data dağılımının homojenliği vb. birçok faktör göz önüne alınmalıdır. Ayda bir kere çalışan, çok büyük raporlar için statik sql ifadeleri yarardan çok zarar getirebilir, çünkü ilk çalıştırılan değerlere göre execution plan hesaplanır. Ama her dakika çalıştırılan ifadeler için de dinamik bir kod yazmak tehlikelidir; diğer planların bellekteki çabuk atılmasına yol açar ve parse CPU gücü gerektirir. Ne yazık ki nerede ne yazılacağını belirleyebileceğimiz altın bir kural yok. Duruma göre gözlem yapıp, en uygun ifadeyi belirlemek ve testlerle kontrol etmek gerekiyor.