BLM 3780 HW1

Hazırlayanlar

Ömer Diner 20011017 omer.diner@std.yildiz.edu.tr

Talha Çelik 21011036 talha.celik1@std.yildiz.edu.tr

PSQL VERSION 16.0

1.SORU

(a) data+indexes loading time :

- You may as well use \timing service in psql.
- For Test1, add 3 distinct latencies, "dataload latency"+"a_idx latency"+ "b_idx latency".

	Loading Latency
Test 1	4.233+1.692+1.601= 7.526 sn
Test 2	19.505 sn

```
test=# INSERT INTO numbers(id,a,b) SELECT floor(random()*1000000+1),floor(random()*1000000+1),floor(random()*1000+1) FROM generate_s
eries(1,2000000);
INSERT 0 2000000
Time: 4233,379 ms (00:04,233)
test=# CREATE INDEX idx_a ON numbers(a);
CREATE INDEX
Time: 1692,468 ms (00:01,692)
test=# CREATE INDEX idx_b ON numbers(b);
CREATE INDEX
Time: 1600,540 ms (00:01,601)
```

Test 1

```
test=# INSERT INTO numbers(id,a,b) SELECT floor(random()*1000000+1),floor(random()*1000000+1),floor(random()*1000+1) FROM generate_s
eries(1,2000000);
INSERT 0 2000000
Time: 19505,309 ms (00:19,505)
```

Test 2

Yorum: Önce index oluşturup sonra yüklü miktarda veri ekleyince her seferinde index üzerinde yapılan düzenlemeler toplam zamanı artırıyor. Var olan veri üzerine index atmak ise veritabanı kendi içinde çeşitli düzenlemeler yapıp indexi daha az oynamayla sıralı bir biçimde oluşturduğu için daha efektif bir yöntem.

(b) index sizes on disk of "numbers" table & a_idx and b_idx:

- both from each Test above and both a-idx and b_idx. Write your comments if they differ between tests or between 2 indexes.

	numbers heap file(w/o idx)	a_idx	b_idx
Test 1	84.46 MB	32 MB	14 MB
Test 2	84.46 MB	39 MB	16 MB

Yorum:

- A sütunu yüksek cardinalitye sahip olduğu için a üzerindeki index boyutu b üzerindeki index boyutundan iki test için de daha fazladır.
- Tablo boyutu iki test için de toplam satır sayısı ve sütun tanımları aynı olduğundan eşittir .

İndex boyutlarını bulmak için kullanılan sorgu:

select indexname, pg_size_pretty(pg_relation_size(indexname::regclass)) as size from pg_indexes where tablename = 'numbers';

Tablo boyutu için pgadminde üstte bulunan statistics sekmesi incelendi.

Örneğin test 1 için karşılaşılan görüntü:

Table size	84.46 MB
Toast table size	
Indexes size	45.79 MB

(c) index statistics. ("space utilization" and "tree height"?)

	a_idx	b_idx
Test 1	3	3
Test 2	3	3

```
SELECT * FROM pgstatindex('idx_a');
```

Output Messages Notifications

version integer	tree_level integer	index_size bigint	root_block_no bigint	internal_pages bigint	leaf_pages bigint	empty_pages bigint	deleted_pages bigint	avg_leaf_density double precision	leaf_fragmentation double precision
4	2	33669120	290	16	4093	0	0	90.14	0

Test 1 a_idx için çalışmanın yapılış şekline örnek

(d) Based on the previous evaluations, which scenario is preferable?
First load data table and then define&load indexes? OR define table&indexes and then load data?

YORUM

a şıkkında görülen zaman farkını temel alarak önce verileri yükleyip sonrasında ise indexleri oluşturmak performans açısından daha iyi ve tercih edilebilir. Ayrıca indexlerin kapladığı yer açısından da bu yöntem daha faydalı sonuç verdi.

2.SORU

"**analyze**" is an admin command in databases to refresh the table stats.
(a) **When** and **why** do we use this command. **What** operations are done internally? Explain briefly. (at most 100 words)

ANALYZE komutu, veritabanında özellikle önemli veri değişikliklerinden(toplu eklemeler, güncellemeler, silmeler) sonra tablo istatistiklerini yenilemek için kullanılır. ANALYZE, veritabanındaki tabloların içeriği hakkında istatistikleri toplar ve sonuçları pg_statistic sistem kataloğunda saklar. Sorgu planlayıcısı, sorgular için en verimli yürütme planlarını belirlemek için bu istatistikleri kullanır. Veri dağılımı hakkındaki bilgileri güncelleyerek sorgu performansını optimize etmeye yardımcı olur. Verileri örnekleme, minimum ve maksimum değerleri alma, distinct değerlerin sayısı gibi her sütundaki veri değerlerinin dağılımı hakkında bilgi toplar ve istatistikleri günceller.

(b) Load the previously defined "numbers" table with 1 M tuples again.

- Test 1: First load and then display the statistics for each attribute (i.e. number of distinct tuples, most_common_values&frequencies)
- Test 2: First load & **analyze** and then display the statistics for each attribute (i.e. number of distinct tuples, most_common_values&frequencies)

Write the meaning of values of the stat's output briefly.
(you may as well use pg_stats utility for the statistics.)

Test1:

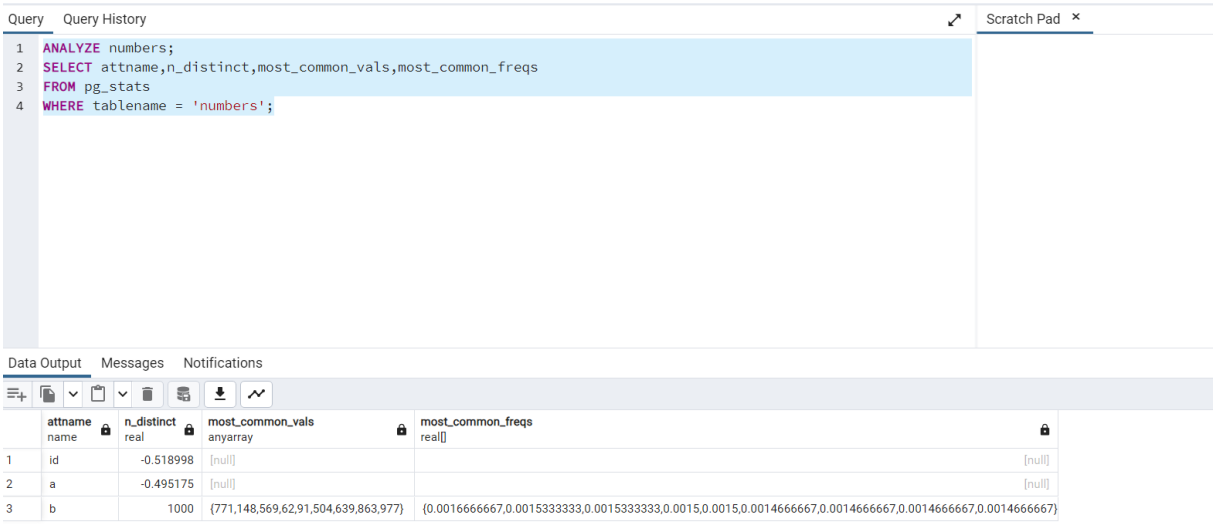
```
SELECT attname,n_distinct,most_common_vals,most_common_freqs
FROM pg_stats
WHERE tablename = 'numbers';
```

Query | Query History | Scratch Pad

Data Output | Messages | Notifications

	attname name	n_distinct real	most_common_vals anyarray	most_common_freqs real[]
1	id	-0.510927	[null]	[null]
2	a	-0.512967	[null]	[null]
3	b	1000	{595,777,772,925,393,357,518,177,481}	{0.0017666667,0.0017,0.0016666667,0.0016333334,0.0015666666,0.0015,0.0015,0.0014666667,0.0014666667}

Test2:



```
ANALYZE numbers;
SELECT attname,n_distinct,most_common_vals,most_common_freqs
FROM pg_stats
WHERE tablename = 'numbers';
```

Query | Query History | Scratch Pad

Data Output | Messages | Notifications

	attname name	n_distinct real	most_common_vals anyarray	most_common_freqs real[]
1	id	-0.518998	[null]	[null]
2	a	-0.495175	[null]	[null]
3	b	1000	{771,148,569,62,91,504,639,863,977}	{0.0016666667,0.0015333333,0.0015333333,0.0015,0.0015,0.0014666667,0.0014666667,0.0014666667,0.0014666667}

YORUM

n_distinct = Sıfırdan büyükse, sütundaki tahmini farklı değer sayısını verir. Sıfırdan küçükse, farklı değer sayısının satır sayısına bölümünün negatifini verir.

Her iki test için de ilk iki sütunda 1 ile 1 milyon arasında rastgele değerler bulunması ama tablonun 2 milyon satır içermesi nedeniyle 1/2'ye yakınsayan değer gözüküyor. B sütunu ise toplam 1000 farklı değere sahip.

most_common_vals = null ise baskın bir değer olmadığı anlamına gelir. **Cardinality yüksek olunca id ve a sütunları için böyle bir sonuç gözükmüş. B sütunu için ise 1 ile 1000 arasındaki sayılardan tablolarda en çok geçen değerler sonuç olarak gösterilmiş.**

most_common_freqs = most_common_vals'daki değerlerin geçme adetleri / toplam satır sayısı

3.SORU

Load the previously defined “numbers” table with 1 M tuples again. Load a-idx & b_idx as well.
We want to display tuples (with all attributes) sorted by “val”. **tabloda "val" olmadığı için "id" olarak aldım.**
(a) Run the query & “explain” the explain output.

```
test=# EXPLAIN ANALYZE SELECT * FROM numbers ORDER BY id;
                                                                  QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=132154.34..134654.34 rows=1000000 width=12) (actual time=632.489..721.854 rows=1000000 loops=1)
   Sort Key: id
   Sort Method: external merge  Disk: 21520kB
   ->  Seq Scan on numbers  (cost=0.00..15406.00 rows=1000000 width=12) (actual time=0.020..102.080 rows=1000000 loops=1)
 Planning Time: 0.131 ms
 Execution Time: 762.840 ms
(6 rows)
```

Explain output

AÇIKLAMA

id üzerinde index olmadığı için sequential scan kullanılarak tüm dosya taranıyor. Rows tablonun satır sayısını gösteriyor. External merge sort yapılıyor çünkü veritabanının sıralama vb. işleri için kullanılan default buffer alanı in-memory bir quick sort yapmaya hazır değil. İşlenen tablo main memorye sığmıyor.

External merge sort = external merge sort karma bir sıralama-birleştirme stratejisi kullanır. Sıralama aşamasında, ana belleğe sığacak kadar küçük veri parçaları okunur, sıralanır ve geçici bir dosyaya yazılır. Birleştirme aşamasında, sıralanan alt dosyalar tek bir büyük dosyada birleştirilir.

(b) Most probably you are seeing **external sorting** !. Now increase “required buffer area” until you see “quick-sorting in main memory”.
If you are already seeing **quick-sorting**, decrease required buffer area until you see “external-sorting in main memory”.

AÇIKLAMA

Default olarak 4MB bulunan work_mem parametresini yavaş yavaş arttırıldı ve 63 MB yapıldıktan sonra sıralama algoritması olarak quick sortun kullanıldığını gözlemlendi.

```
test=# show work_mem;
 work_mem
----------
 61MB
(1 row)


test=# EXPLAIN ANALYZE SELECT * FROM numbers ORDER BY id;
                                                                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=115063.84..117563.84 rows=1000000 width=12) (actual time=466.439..551.608 rows=1000000 loops=1)
   Sort Key: id
   Sort Method: external merge  Disk: 21512kB
   ->  Seq Scan on numbers  (cost=0.00..15406.00 rows=1000000 width=12) (actual time=0.012..68.415 rows=1000000 loops=1)
 Planning Time: 0.064 ms
 Execution Time: 581.970 ms
(6 rows)
```

61 MB external merge sort

```
test=# show work_mem;
 work_mem
----------
 63MB
(1 row)


test=# EXPLAIN ANALYZE SELECT * FROM numbers ORDER BY id;
                                                                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------------------------------------
 Sort  (cost=115063.84..117563.84 rows=1000000 width=12) (actual time=327.928..453.834 rows=1000000 loops=1)
   Sort Key: id
   Sort Method: quicksort  Memory: 63255kB
   ->  Seq Scan on numbers  (cost=0.00..15406.00 rows=1000000 width=12) (actual time=0.017..74.150 rows=1000000 loops=1)
 Planning Time: 2.175 ms
 Execution Time: 483.875 ms
(6 rows)


test=#
```

63 MB olunca quicksort

(c) Have you experinced better execution times when you increased the buffer size? Why or why not?

AÇIKLAMA

Parametreyi artırdıkça hız arttı çünkü daha geniş bir alana tek seferde daha çok veri getiriliyor bu da diske erişim açısından maaliyeti düşürüyor. Özellikle alanı iyice arttırıp in-memory quicksort yapabilmesine yetecek kapasiteyi veritabanına sağlarsak execution time çok hızlanıyor.

4.SORU

(4)

Load the previously defined "numbers" table with 1 M tuples again. Load a-idx & b_idx as well.

(a) **We want to count distinct "a" values.** Run the query & "explain" the explain output. Why does it use (or not use) idx ?

```
test=# EXPLAIN SELECT COUNT(DISTINCT a) FROM numbers;
                                  QUERY PLAN
-------------------------------------------------------------------------------------------
 Aggregate  (cost=27132.42..27132.43 rows=1 width=8)
   ->  Index Only Scan using idx_a on numbers  (cost=0.42..24632.42 rows=1000000 width=4)
(2 rows)
```

AÇIKLAMA

Burada index only scan kullanılması uygun görülmüş yani index kullanılıyor çünkü query'de sadece a sütunu geçiyor ve a sütunu üzerinde index bulunuyor. İndex-only olması sebebiyle ana tabloya fiziksel olarak erişim işlemi yapılmıyor sadece a üzerinde bulunan index üzerinden istenilen işlem gerçekleştiriliyor. Bu yöntem, tablonun heap dosyasına erişip tarama yapmaktan daha verimli görülmüş.

İndex-only scan = Belirli sorgu türlerinin tablolardan değil, yalnızca index üzerinden veri alınarak karşılanmasına olanak tanır. Bu, sorguları çalıştırmak için gereken i/o miktarında önemli bir azalmaya neden olur.

(b) **We want to list all tuples (with all attributes) having a B-value higher than 700.** Run the query & "explain" the explain output. Why does it use (or not use) idx ? What is the threshold value to see sequential file scan?

```
test=# explain select * from numbers where b>700;
                                QUERY PLAN
-------------------------------------------------------------------------
 Bitmap Heap Scan on numbers  (cost=3362.63..12488.32 rows=297575 width=12)
   Recheck Cond: (b > 700)
   ->  Bitmap Index Scan on idx_b  (cost=0.00..3288.24 rows=297575 width=0)
         Index Cond: (b > 700)
(4 rows)
```

AÇIKLAMA

Where koşulunda bulunan B üstünde index bulunduğu için ve aynı zamanda istenen koşul tablonun küçük bir bölümüne denk geldiği için var olan index kullanılmış.

Planlayıcı, sorgu sequential scan gibi toplu okumanın avantajlarından yararlanabilecek kadar büyük miktarda veri istediğinde, ancak bu veri aslında tüm tablonun işlenmesini gerektirecek kadar büyük olmadığından bitmap index tarama yöntemini seçmiş. Bitmap index scan, sequential ve index scan arasında bir şey olarak düşünülebilir. Bitmap index scan, her zaman bir Bitmap heap scan ile birlikte çalışır; birincisi tüm uygun satır konumlarını bulmak için dizini tarar ve bir bitmap oluşturur, ardından ikincisi yığın sayfalarını tek tek taramak ve satırları toplamak için bu bitmapi kullanır.

Threshold value = yapılan testlerde b>x koşulunda x'in 482 ve altında olduğunda planlayıcının indexi kullanmayı bırakarak sequential scan kullanmaya yöneldiği gözlemlendi. Çünkü b sütunu 1 ile 1000 arasında değerler alıyordu. Dağılımın her değer için eşit olacağını düşünürsek verilerin yarısında b değeri 500'den küçük diğer yarısında 500'den büyüktür. Tamamen eşit dağılım olmadığı için testte bulunan 482 değeri de bunu destekler. Satırların yarısından fazlasının karşıladığı bir koşul için index kullanmak verimsiz olur, seek artar. Dosya taraması bu durum özelinde daha verimli.

```
test=# explain select * from numbers where b>480;
                          QUERY PLAN
---------------------------------------------------------------
 Seq Scan on numbers  (cost=0.00..17906.00 rows=519770 width=12)
   Filter: (b > 480)
(2 rows)


test=# explain select * from numbers where b>485;
                                QUERY PLAN
---------------------------------------------------------------------------
 Bitmap Heap Scan on numbers  (cost=5818.50..17660.10 rows=514848 width=12)
   Recheck Cond: (b > 485)
   ->  Bitmap Index Scan on idx_b  (cost=0.00..5689.78 rows=514848 width=0)
         Index Cond: (b > 485)
(4 rows)


test=# explain select * from numbers where b>483;
                                QUERY PLAN
---------------------------------------------------------------------------
 Bitmap Heap Scan on numbers  (cost=5841.76..17707.97 rows=516817 width=12)
   Recheck Cond: (b > 483)
   ->  Bitmap Index Scan on idx_b  (cost=0.00..5712.55 rows=516817 width=0)
         Index Cond: (b > 483)
(4 rows)


test=# explain select * from numbers where b>482;;
                          QUERY PLAN
---------------------------------------------------------------
 Seq Scan on numbers  (cost=0.00..17906.00 rows=517801 width=12)
   Filter: (b > 482)
(2 rows)
```

Threshold değerini bulmak için yapılan denemeler

(c) **We want to count the total number of A-values that are equal to B-values in "any tuple in the table", including duplicates.** Run the query & "explain" the explain output. Why does it use (or not use) idx ?

```
test=# explain  select count(*) from numbers n1,numbers n2 where n1.a=n2.b;
                                                        QUERY PLAN
---------------------------------------------------------------------------------------------------------------------------
 Finalize Aggregate  (cost=25288.95..25288.96 rows=1 width=8)
   ->  Gather  (cost=25288.74..25288.95 rows=2 width=8)
         Workers Planned: 2
         ->  Partial Aggregate  (cost=24288.74..24288.75 rows=1 width=8)
               ->  Merge Join  (cost=501.82..22187.59 rows=840457 width=0)
                     Merge Cond: (n2.b = n1.a)
                     ->  Parallel Index Only Scan using idx_b on numbers n2  (cost=0.42..12715.09 rows=416667 width=4)
                     ->  Index Only Scan using idx_a on numbers n1  (cost=0.42..24632.42 rows=1000000 width=4)
(8 rows)
```

AÇIKLAMA

A ve b değerlerine index üzerinden erişebildiğimiz için index only scan ile ikisi de toplanmış daha sonra a=b koşulunu sağlayanlar merge edilmiş. Partial aggregate ile her worker için count işlemi yapılmış daha sonra gather ile sonuçlar birleştirilmiş.

İndex kullanılmış çünkü count ile toplam sayıyı istediğimiz için ana tabloya uğramaya gerek kalmadan indexler kullanılarak koşul karşılaştırılabilir, ana tabloya gidip ekstra bir sütuna erişmeye gerek yok.