1.soru

Öncelikle ödev boyunca bu ve bundan sonraki bütün soruları pgadmin kullanarak arayüzünde sql sorgularıyla yaptım. İstenilen tabloyu pgadmin kullanarak sql sorgusu ile oluşturdum. Daha sonra b niteliği 0 ve 1.5 milyon arası rastgele değerler alan 2 milyon veriyi generate_series() yöntemi ile tabloya insert ettim. Tablonun kaç mb olduğunu sorguladım ve 100 mb olarak buldum. İçerdiği disk sayfa sayısını bulmak için bir hesap yaptım. Default olarak postgresql'de page boyutu 8KB'dır bu yüzden tablo boyutunu 8 KB'a böldüm ve tablonun 12739 tane page olarak tutulduğunu buldum. Daha sonra sql sorgusuyla b sütununa ait en küçük değeri, en büyük değeri, ortalamayı ve standart sapması olmak üzere istatistiklerini yazdırdım. Ve son olarak da pg_attribute sistem kataloğunu sorgulayarak tabloda yer alan niteliklerin tipini, değişken uzunluklu olup olmadığını ve kaç bayt olarak tutulduğunu sorguladım. Bunun sonucunda varchar olan a niteliğinin değişken uzunluklu b ve c niteliklerinin sabit uzunluklu olduğunu çıkardım.

2.soru

Soruda istendiği gibi b niteliği üzerinde indeks oluşturmayı /timing komutu ile denedim ancak pgadmin'de bu komut tanınmadığından başka bir yöntem olan postgresql'in built-in fonksiyonlarını kullandım. Böylece süreyi indeks oluşturmadan çekiyorum ve arada indeks oluşturuyorum ve oluşma bittiği anda time_stamp'tan güncel zamanı çekiyorum. İkisini birbirinden çıkarınca da indeks oluşturma için geçen zamanı bulmuş oluyorum. Bu süreyi birkaç defa silip yeniden deneyerek yaklaşık olarak 1.3 sn olarak buldum. Daha sonra yeni bir sorguyla oluşturduğum indeksin boyutunu 36 mb olarak öğrendim ve oluşan B+-Tree yapısının totalde 4568 page olarak tutulduğunu gördüm. B+-Tree'nin her seviyesindeki sayfa sayısını ve boyutunu öğrenmeyi çok araştırdım fakat bu bilgiyi elde edecek herhangi bir sorgu bulamadım. Daha sonra indeksin kullanımıyla alakalı ayrıntılı bir sorgu daha gerçekleştirdim. Burada da indeksin şimdiye dek kaç defa kullanıldığını ve bu kullanmalarda kaç kayıt okunduğu gibi bilgileri elde ettim. Daha sonra tablo üzerinden sql sorgusuyla b niteliği üzerinde kaç değerin tekrar ettiğini, bunların toplamda kaç kayıt üzerinde olduğunu ve bir değer için max tekrar miktarını buldum. ..

3.soru

Postgresql de default fillfactor değeri olan 90'ı 60 olarak değiştirerek yeni bir indeks oluşturdum.Bunun sonucunda B+-Tree'nin içerdiği disk sayfa sayısının 6872 olarak arttığını gözlemledim. Bu beklediğim bir değerdi çünkü her bir node'da artık daha az record tutulacağı için ağaçtaki node sayısı yani sayfa sayısı artacaktı. Dolayısıyla buna bağlı olarak da indeksin boyutu da artıp 54 mb oldu. Böyle bir indeksi şu yüzden isteyebiliriz. Yoğun ve sık veri ekleme'nin gerek olacağı durumlarda bu indeks faydalı olur çünkü halihazırda node'larda boş yerler daha fazla olacağından yeni eklenen veri direkt buraya yerleşir bu da bir önceki indeks gibi ağacın yapısının sürekli değişmesine neden olmaz. Çünkü node'da yer olmadığında o node'u ikiye bölüp bu değişikliklerin root'a doğru yansıtılması illaki bir overhead durumu oluşturacaktır. Yani yer verimliliği düşük ağaç oluşturarak belki ilk başta fazladan yer tutup

bir zarara giriyoruz ancak bu uzun vadede bize çok veri ekleme olacaksa bir kazanç olarak geri dönüyor.

4.soru

Öncelikle t_yogun isminde yeni bir tablo oluşturup içine de b niteliği 500-600 bin arasında yoğunlaşan 2 milyon veri ekledim. Generate_series() yöntemi ile oluştururken 2'ye mod alarak her 2 kaydın birinin 500-600 bin arasına düşmesini sağladım. Daha sonra bu tablo üzerinde b niteliğini search-key alarak bir indeks yapısı oluşturdum ve indeksin boyunun daha önce oluşturduğumuz iki indekse göre daha düşük olarak 28 mb buldum. Diskteki sayfa sayısı da azalarak 3549 oldu. Bunun sebebi şudur: sıkışık bir verisetinde tekrar eden değerler yoğunlaştığı için aynı search-key değerine sahip kayıtların tek bir leaf'te gruplanması sağlanır. Bu sayede toplam farklı search-key sayısı azaldı dolayısıyla ağaç küçüldü boyutu ve diskteki sayfa sayısı da azalmış oldu. Bunun şu şekilde denemesini de yaptım. Bu tabloyu birkaç kez silip yeniden oluşturdum ve sırasıyla önce her 3 kayıtta bir daha sonra 4 kayıtta bir 500-600 bin arasına düşen bir veriseti oluşturdum. Ve beklenildiği gibi yoğunluk azaldıkça indeksin boyutu ve sayfa sayısı arttı.

5.soru

Postgresql default sayfa boyutu = 8KB, search-key boyutu= 4 bayt, indeks boyutu = 8 bayt

$Max\ Order = \frac{1024*8}{4+8} = 682$ olarak buluyoruz. Bu bir düğümde en fazla kaç indeks kaydının tutulacağını gösteriyor. Fillfactor değeri ile bu orderin oranını ayarlıyoruz. Biz 2. Soru için hesap yapacağımızdan fillfactor=90 olarak varsayılan olacak. $order = 682 * \frac{90}{100} = 612$ olarak bulunur. 2 milyon kaydımız olduğu için ağaç yüksekliğini $\boldsymbol{yükseklik} = \log_{612} 2000000 \cong 2.26$ olarak buluruz bu da ağacımız yüksekliğinin 3 olması gerektiğini belirtir.

6.soru

İndeksi oluştururken b niteliği üzerine değer aralığı [0,1.5 milyon] olarak oluşturuyoruz. Ancak burda rastgele oluşum olduğu için her seferinde farklı değerler gelebilir ve indeks de bu değerlere göre oluşuyor. Bu yüzden önce bir sql sorgusu ile en büyük ve en küçük b değerlerini buluyoruz. Bu bulduğumuz değerler en düşük ve en yüksek search-key oluyor. Örneğin benim verilerimde en küçük 0 en büyük de 1.499.497 oluyor. Yani bu 1.499.497 den sonra herhangi bir indeksleme yok ve kullanıcı o aralıktan herhangi bir sorgu yaparsa bunun için indeks kullanamayacağız. Bu yüzden benim verilerim için bu sorunun cevabı

1499497-0 = 1499497 oluyor.

7.soru

Sorgu sonucunda COUNT(DISTINCT b) işleminin gerçekleştiği AGGREGATE kısmının analizinde bu sorgunun çok fazla bir ek iş yükü oluşturmadığı görülüyor.

“Index Only Scan using idx_b on t” kısmı ise postgresql’in yalnızca indeksi kullanarak sorguyu çözümlediğini gösteriyor.

Postgresql B+-Tree indeksini kullanarak hızlı bir şekilde benzersiz b değerlerini elde ediyor ve bu sıralı yapıya sahip bir indeksin verimli bir şekilde kullanılmasından kaynaklanıyor.

8.soru

Sorgu sonucunun ilk aşamasındaki ‘GroupAggregate’ kısmı sorgunun GROUP BY kısmını temsil ediyor. Burdaki cost değerinin başlangıç ve tamamlanma maliyeti arasındaki ciddi fark ise sorgunun büyük bir veri kümesi üzerinde çalıştığını ve grup bazında hesaplama maliyetinin çok fazla olduğunu gösteriyor denilebilir.

Sorgu sonucunun ‘Index Scan using idx_b on t’ kısmı ise ‘idx_b’ indeksinin bu sorguda kullanıldığını gösteriyor ancak bu indeks b sütunu üzerine olduğu sadece GROUP BY B kısmında fayda sağlıyor. Onun haricinde gruplanan veriler üzerinde bir de herhangi bir indekse sahip olmayan a ve c attributeları üzerinde de bir WHERE işlemi gerçekleşeceğinden ve halihazırdaki idx_b indeksi bu kontrol işlemine herhangi bir fayda sağlamayacağından aslında sorgu sadece indeks kullanıp çok hızlı bir şekilde çalışıyor denilemez. Çünkü her bir veri için a ve c okuması yapmak zorunda kalacağız.

9.soru

Halihazırdaki T tablosundan a sütunundaki verileri sıralayarak yeni bir t_sorted tablosu oluşturma kodunu analyze ederek çalıştırdığımızda şu sonuçlara rastlıyoruz:

Tablo oluşturulurken external merge sort kullanılıyor çünkü veri memory’de sıralanamayacak kadar büyük bu yüzden önce diske yazılıyor burada veri küçük parçalara ayrılıp önce kendi içinde sıralanıp sonra merge ediliyor.

‘k’ değeri yapılan merge işleminde aynı anda kaç tane sıralanan parçanın birleştirileceğini ifade ediyor.

Yaptığımız sorgu sonucunda default olarak work_mem değerinin 4 mb olduğunu görüyoruz. Ve tablomuzdaki verinin boyutu ise 100mb. Bu halde 100/4 = 25 olarak mevcut hafızada gerektirdiği iterasyon sayısını bulabiliriz.

Harici sıralama işlemi, verilerin disk üzerinde sıralanıp okunmasını gerektiren bir süreç olduğu için gecikme internal sort işlemine göre daha fazla olmuştur.

Sıralama işleminin 2 iş parçacığı ile paralel olarak yapılması işlem süresini kısalmıştır.

2 iş parçacığı da farklı miktarda disk alanı kullanmış.

Video linki : https://youtu.be/v6s5ManAqLI

Yusuf Safa Köksal 21011002