# BLM 5117 - Veri Tabanı Sistemlerinin Gerçeklenmesi Ödev -1

Metin Uslu - 235B7014

## Ön Bilgi & Hazırlık

**DB:** [PostgreSQL 17.0](https://www.postgresql.org/about/news/postgresql-17-released-2936/)

**Client:** [pgAdmin](https://www.pgadmin.org/), [DBeaver](https://dbeaver.io/)

**Installation:** [Docker Compose](https://docs.docker.com/compose/) (Docker Compose version v2.29.7-desktop.1)

**GitHub Repo:** <https://github.com/metinuslu/blm5117_dbms_hw1>

**Not:** Aşağıdaki çıktıları docker-compose.yaml ve sql scriptleri paylaşılarak tekrarlanabilir bir şekilde yapılabilmesini sağlanmaktadır. Sadece random fonksiyonu neticesinde oluşan veri seti içerisinde değişiklikler göz ardı edilmemelidir.

**>** docker exec -it postgres bash

**>** psql -h localhost -d postgres -U db\_user

-- Show DB Version

**SELECT version**();

PostgreSQL 17.0 (Debian 17.0-1.pgdg120+1) on x86\_64-pc-linux-gnu, compiled by gcc (Debian 12.2.0-14) 12.2.0, 64-bit

**PostgreSQL Sistem Fonksiyonları (Query’lerde Kullanılan Fonksiyonlar)**

* **pg\_size\_pretty**: PostgreSQL'de, bir boyut değerini (bayt cinsinden) daha okunabilir (human readable) bir biçimde sunmak için kullanılan bir sistem fonksiyonudur. Bu fonksiyon, bayt cinsinden verilen bir değeri KB, MB, GB gibi daha anlaşılır birimlerde gösterir.
* **pg\_table\_size:** Bir tablonun yalnızca verilerinin ve tablo üzerindeki TOAST verilerinin boyutunu döner (indeksler dahil edilmez).
* **pg\_total\_relation\_size:** Bu fonksiyon, belirtilen tablo için toplam disk alanını bayt cinsinden döner.
* **pg\_stats:** PostgreSQL'in otomatik olarak topladığı istatistiksel bilgileri tutan bir sistem görünümüdür.
* **pg\_class:** PostgreSQL’de tablolar, dizinler (indeksler), görünümler ve diğer ilişkisel yapılar hakkında genel bilgiler içerir.
* **pg\_indexes\_size:** Bir tablonun tüm indekslerinin toplam boyutunu döner.
* **pg\_relation\_size:** Bir tablonun yalnızca temel veri dosyasının (indeks ve TOAST hariç) boyutunu döner.
* **pg\_stat\_all\_indexes:** Bir tablo üzerindeki tüm indeksler için kullanım istatistiklerini döner. İndekslerin kaç kez kullanıldığı, toplamda kaç satır okunduğu gibi bilgileri içerir.
* **pg\_indexes:** PostgreSQL'deki tüm indekslerin isimlerini, tanımlarını ve ilgili tablolarını döner. İndeks yapısını anlamak için faydalıdır.
* **pgstatindex:** Belirli bir indeksin yapısı ve durumu hakkında bilgi döner. Bu bilgi, pageinspect uzantısı yüklendiğinde kullanılabilir. İndeksin doluluk oranı, yaprak sayfa sayısı gibi veriler içerir.

## Ödev Ön Bilgi

Şeması T(a varchar(40), b int, c bool) olan HEAP dosyası 2 milyon kayıt içersin. b-niteliği değer aralığı [0,1.5 milyon] olmak üzere; searchkey= b niteliği üzerinde bir B+-tree indeks düşünelim. Böylece yapraklarda (rid, searchkey=b olan kaydın rid’si olmak üzere) ikilileri saklanıyor.

## Problem-1

**Açıklama:** Bahsedilen tabloyu üretin (hariçte python sonra COPY komut ile, veya dahilde generate\_series() yöntemleri ile bunu yapabilirsiniz). Ortaya çıkan T tablosunun büyüklüğü (MB), içerdiği disk sayfa sayısı, ve b niteliğine ait sütun istatistiklerini inceleyin (Bunlar için sistem görüntü fonksiyonlarını kullanabilirisiniz). pg\_attribute sistem kataloğunu da sorgulayarak dosyanın sabit / değişken uzunluklu kayıt dosyası olma durumunu öğrenin.

* Burada DDL komutlarından Create Database ile vtsg\_db oluşturulmuş ardından \c vtsg\_db ile Database’e bağlantı kurulmuş oluşturulmaktadır.

-- Connect DB

psql -h localhost -d postgres -U db\_user

-- Create Database and Schema

CREATE DATABASE vtsg\_db;

-- List DBs

\l

-- Connect Database

\c vtsg\_db;

* Burada DDL komutlarından Create Table komutu kullanılarak A, B ve C kolon adları ile Varchar, Int ve Bool tipinde T tablosu oluşturulmaktadır.

-- Create Table

**CREATE** **TABLE** vtsg\_db.public.T(

a **VARCHAR**(40),

b INT,

c **BOOLEAN**

);

* Burada 2 milyon adet a(VarChar), b(Int ve 0 ile 1.5 Milyon) ve c(Bool) kolonları Postgre built-in ***generate\_series*** fonksiyonu ile t tablosu doldurulmaktadır. Ben burada a kolonun 5 karakterden oluşmasını ve ilk karakterin alfabetik olması,

B kolonu koşulu ile oluşturdum.

-- Generate and Insert 2 Million Random Values

**INSERT** **INTO** vtsg\_db.public.T (a, b, c)

**select**

**chr**(**trunc**(65 + **random**() \* 25)::**int**) || **upper**(**substring**(**md5**(**random**()::**text**) **FROM** 1 **FOR** 4)), --A Kolonu: İlk karakter Alfabetik ardından 4 karakter Alfanümerik olmak üzere 5 Karakter

**trunc**(**random**() \* 1500000)::**int**, -- B Kolonu: 0 ile 1.5 milyona kadar

**random**() > 0.5 -- C Kolonu: Rastgele TRUE veya FALSE

**FROM**

**generate\_series**(1, 2000000);

* Burada tablonun count bilgisi alınarak kontrol edilmektedir.

-- Control Table Count

**select** **COUNT**(\*) **as** *CNT* **from** vtsg\_db.public.T;

* Burada tablo üzerinde random 10 satır bilgisi getirilmektedir.

-- Show Table Rows only 10 rows

**select** \* **from** vtsg\_db.public.T **limit** 10;

* Burada tablonun metadata bilgileri gösterilmektedir.

-- Control Table Metadata Info

**SELECT**

--column\_name, data\_type

\*

**FROM** information\_schema.**columns**

**WHERE** ***table\_catalog***=**'vtsg\_db'** **and** ***table\_schema*** = **'public'** **and** ***table\_name*** = **'t'**;

### T Tablosunun Büyüklüğü (MB)

* Burada tablonun büyüklük (table size) bilgisi **pg\_table\_size** fonksiyonu ile alınır ve **pg\_size\_pretty** fonksiyonu ile daha okunur hale getirilir. Tablonun 85 MB (86536 Byte) büyüklüğünde olduğunu görülmektedir.

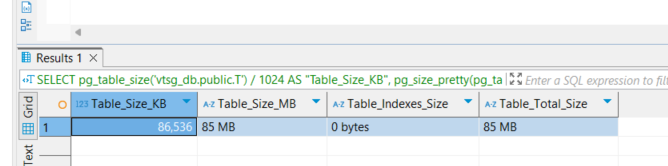
**SELECT**

**pg\_table\_size**('vtsg\_db.public.T') / 1024 **AS** *"Table\_Size\_KB"*, --Tablonun yalnızca veri boyutu (sadece tablo verisi) Byte->KiloByte olarak verir

**pg\_size\_pretty**(**pg\_table\_size**('vtsg\_db.public.T')) **AS** *"Table\_Size\_MB"*, --Tablonun yalnızca veri boyutu (sadece tablo verisi)

**pg\_size\_pretty**(**pg\_indexes\_size**('vtsg\_db.public.T')) **AS** *"Table\_Indexes\_Size"*, -- Tablonun yalnızca indeks boyutu

**pg\_size\_pretty**(**pg\_total\_relation\_size**('vtsg\_db.public.T')) **AS** *"Table\_Total\_Size"* --Tablonun toplam boyutu (veri, indeks ve boş alan dahil)



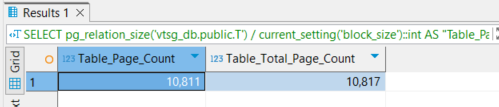
### T Tablosunun İçerdiği Disk Sayfa Sayısı

* Burada tablonun içerdiğini disk sayfa sayısı bilgisi **pg\_relation\_size** fonksiyonu ile byte olarak alınır ve Block Size: 8192 (Byte 🡪 Kilo Byte dönüşümü) bölünür. Bu çıktı, T tablosunun toplamda 10,811 sayfa içerdiğini gösterir. Her sayfa genellikle 8 KB olduğu için, bu sayfa sayısı, toplam disk alanının ne kadarını kapsadığını belirtir. 10,811 \* 8192 (8 KB) = ~88,56 MB olduğunu gösterir.

**SELECT**

**pg\_relation\_size**('vtsg\_db.public.T') / **current\_setting**('block\_size')::**int** **AS** *"Table\_Page\_Count"*, --Table Page Size

**pg\_total\_relation\_size**('vtsg\_db.public.T') / **current\_setting**('block\_size')::**int** **AS** *"Table\_Total\_Page\_Count"* --All T Objects (Table+ Index + Other Datas) Page Size



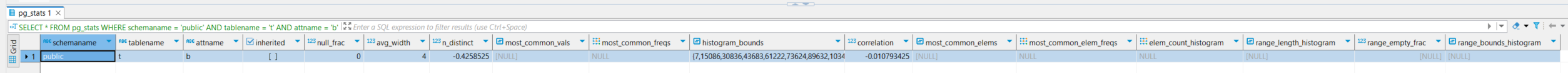
### b Niteliğine Ait Sütun İstatistikleri

* Burada **pg\_stats** fonksiyonu ile b tablosunun istatistikleri getirilmiştir. Çok fazla ist.’ler yer almaktadır. Tablo üzerinde sorgu optimizasyonu açısından buradaki değerler önem arz etmedir. En önemli olanlarından bazıları aşağıda paylaşılmıştır.

**SELECT** \*

**FROM** ***pg\_stats***

**WHERE** ***schemaname*** = **'public'** **AND** ***tablename*** = **'t'** **AND** ***attname*** = **'b'**;



**null\_frac:** Boş Değer Oranı: Tablodaki değeri 0 olması b kolonu içerisinde Null kayıt olmadığını söyleyebiliriz.

**avg\_widht:** Sütun Ort. Veri (Byte) Genişliği: Tablodaki b kolonu Int olduğu için 4 Byte olarak gözükmektedir.

**n\_distinct:** Benzersiz Değer Sayısı: Tabloda b kolonu -0.425825 değerine sahiptir. Burada değerin negatif olması benzersiz değerlerin tam olarak belirlenemediğini göstermektedir.

**most\_common\_vals ve most\_common\_freqs:** En Yaygın Değer ve Sıklığı(Frekansları): Tabloda boş olması tablodaki değerlerin dağılımının çok çeşitli veya belirgin bir şekilde öne çıkan değerin olmadığını gösterir. Yine sık görülen değerlerin olmadığını gösterir.

**correlation:** Korelasyon**:** Sütundaki değerlerin sıralı olup olmadığını söyler. Korelasyonu değeri 1’e yakın ise sıralı olduğunu ifade eder. B kolunu için correlation değeri -0.010793425 olduğu için sıralı bir dağılım göstermediğini, sıralamanın dağınık olduğunu söyleyebiliriz. Korelasyon değeri sorgu performansını etkilemektedir.

### pg\_attribute Sistem Kataloğu ile Dosyanın Sabit / Değişken Uzunluklu Kayıt Dosyası Olma Durumu

**SELECT**

*a*.***attname***,

*a*.***atttypid***::**regtype** **AS** data\_type,

*a*.***attlen***,

**CASE**

**WHEN** *a*.***attlen*** = -1 **THEN** **'Variable Length'**

**WHEN** *a*.***attlen*** > 0 **THEN** **'Fixed Length'**

**ELSE** **'Unknown'**

**END** **AS** *length\_type*

**FROM**

***pg\_attribute*** *a*

**JOIN**

***pg\_class*** *c* **ON** *a*.***attrelid*** = *c*.***oid***

**WHERE**

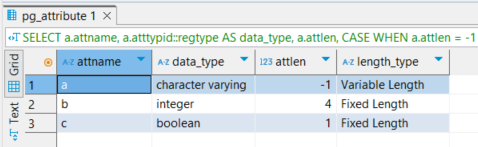
*c*.***relname*** = **'t'** **AND** *a*.***attnum*** > 0;

* PostgreSQL'de bir tablonun sabit veya değişken uzunluklu kayıt dosyası olma durumunu öğrenmek için **pg\_attribute** sistem kataloğunu sorgulayabilirsiniz. Özellikle, bir sütunun veri tipinin sabit veya değişken uzunlukta olup olmadığını anlamak için attlen sütununu kontrol edebilirsiniz. Sabit ve Değişken Uzunluklu Kayıtlar;

**Sabit Uzunluklu Veri Tipleri:** CHAR, INTEGER, BOOLEAN gibi veri tipleri sabit uzunlukta alan kaplar.

**Değişken Uzunluklu Veri Tipleri:** VARCHAR, TEXT, gibi veri tipleri değişken uzunlukta alan kaplar.

Sorgu çıktısında da görüldüğü üzere a kolonu değişken uzunluklu, b ve c kolonları ise sabit uzunluklu olduğu ***attlen*** değerleri üzerinden görülmüştür.



## Problem-2

**Açıklama:** Soruda bahsedilen B+-tree indeksi PgSQL’de oluşturun: Indeks yükleme esnasında \timing komutu ile indeks yükleme gecikmesi değerini ölçün. Ortaya çıkan ağacın yüksekliğini, her seviyedeki sayfa sayısını ve ağacın yüksekliğini, pg\_stat\_all\_indexes sistem görünümünü kullanarak öğrenin, diğer indeks istatistiklerini inceleyin, yorumlayın. Mesela, [0, 1.5 milyon] değer aralığında 2 milyon kayıt ürettiğimiz için indekste tekrar değerler olmalı. Bunların sayısı ne kadar? Bunu ilk olarak T tablosunda bir SQL ile bulun. Sonra sistem katalog sorguları ile bulun. Sistem katalog ile bulduklarınız yanlış mı? O zaman sistem kataloglarını güncelleyip T üzerinde SQL ile bulduklarınızla aynı değerleri elde edin.

### Index Oluşturma ve Ölçme

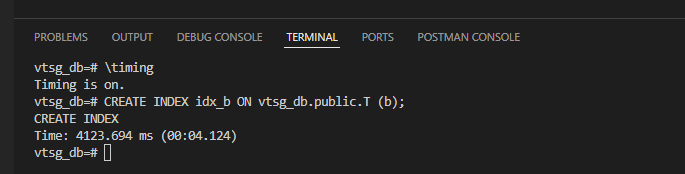
vtsg\_db=# \timing

Timing is on.

vtsg\_db=# CREATE INDEX idx\_b ON vtsg\_db.public.T (b);

CREATE INDEX

Time: 4123.694 ms (00:04.124)



### pg\_stat\_all\_indexes

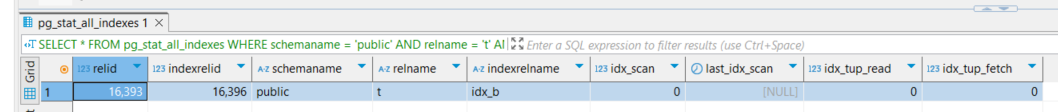
Ortaya çıkan ağacın yüksekliğini, her seviyedeki sayfa sayısını ve ağacın yüksekliğini, pg\_stat\_all\_indexes sistem görünümünü kullanarak öğrenin, diğer indeks istatistiklerini inceleyin, yorumlayın.

* pg\_stat\_all\_indexes ve bt\_metap sistem fonksiyonlarını kullanarak index’in istatistikleri aşağıdaki gösterilmiştir.

**SELECT** \*

**FROM** pg\_stat\_all\_indexes

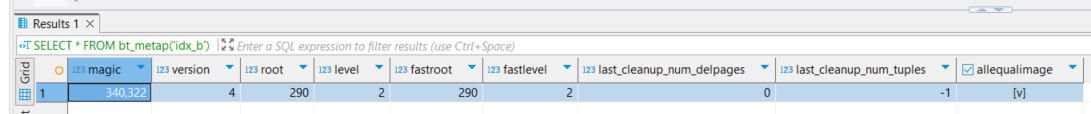
**WHERE** schemaname = 'public' **AND** relname = 't' **AND** indexrelname = 'idx\_b';



|  |  |  |
| --- | --- | --- |
| relid | 16393 | T tablosunun PostgreSQL içindeki benzersiz tanımlayıcı kimliği (Object ID). |
| indexrelid | 16396 | idx\_b indeksinin PostgreSQL içindeki benzersiz tanımlayıcı kimliği (Object ID). |
| schemaname | public | İndeksin ait olduğu şemanın adı (public, PostgreSQL'in varsayılan şeması). |
| relname | t | İndeksin ait olduğu tablonun adı (T). |
| indexrelname | idx\_b | İndeksin adı (idx\_b, b kolonu için oluşturulan B+Tree indeksi). |
| **idx\_scan** | **0** | **İndeksin sorgularda hiç kullanılmadığını gösteriyor.** |
| **last\_idx\_scan** | **NULL** | **İndeks üzerinde bir tarama yapılmadığı için son tarama zamanı kaydedilmemiş.** |
| **idx\_tup\_read** | **0** | **İndeksten hiçbir satırın okunmadığını gösteriyor.** |
| **idx\_tup\_fetch** | **0** | **İndeks kullanılarak hiçbir satırın geri döndürülmediğini gösteriyor.** |

**CREATE** **EXTENSION** **IF** **NOT** **EXISTS** pageinspect;

**SELECT** \* **FROM** bt\_metap('idx\_b');



|  |  |  |
| --- | --- | --- |
| **Sütun Adı** | **Değer** | **Açıklama** |
| magic | 340322 | B+Tree'nin dahili tanımlayıcı değeri. Bu değer, indeksin geçerli bir B+Tree olduğunu doğrulamak için kullanılır. |
| version | 4 | B+Tree sürümü. PostgreSQL, sürüm 4'ü kullanır. Bu, indeksin güncel yapıda olduğunu gösterir. |
| root | 290 | Kök düğümün blok numarası. Bu, ağacın en üst seviyesinde yer alan kök düğümün fiziksel yerini (disk bloğunu) ifade eder. |
| level | 2 | İndeksin yüksekliği. Bu, ağacın kökten yapraklara kadar toplam kaç seviyeden oluştuğunu belirtir. |
| fastroot | 290 | Hızlı erişim için kullanılan kök düğümün blok numarası. Genellikle root ile aynı olur. |
| fastlevel | 2 | Hızlı erişim için kullanılan seviyenin derinliği. Bu da genelde level ile aynıdır. |
| last\_cleanup\_num\_delpages | 0 | Son temizlik işleminde (vacuum/reindex) silinen sayfa sayısı. Bu indeks üzerinde henüz bir temizlik yapılmamış. |
| last\_cleanup\_num\_tuples | -1.0 | Son temizlik işleminde temizlenen toplam tuple sayısı. Bu, temizlik yapılmadığı için -1.0 olarak belirtilmiş. |
| allequalimage | true | Eşit sayfa yapısına sahip olma durumu. Bu, indeksin sayfa yapılarının dengeli olduğunu gösterir (yani eşit dağılım). |

### Tekrar Eden Kayıtlar

Mesela, [0, 1.5 milyon] değer aralığında 2 milyon kayıt ürettiğimiz için indekste tekrar değerler olmalı. Bunların sayısı ne kadar? Bunu ilk olarak T tablosunda bir SQL ile bulun. Sonra sistem katalog sorguları ile bulun. Sistem katalog ile bulduklarınız yanlış mı? O zaman sistem kataloglarını güncelleyip T üzerinde SQL ile bulduklarınızla aynı değerleri elde edin.

#### Query İle Bulunması Tekrar Edenlerin Listelenmesi ve Sayısının Bulunması

--Tekrar Eden Değerlerin Listelenmesi

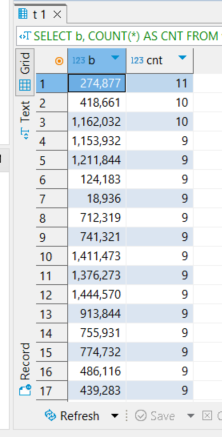
**SELECT** b, **COUNT**(\*) **AS** *CNT*

**FROM** vtsg\_db.public.T

**GROUP** **BY** b

**HAVING** **COUNT**(\*) > 1

**ORDER** **BY** 2 **DESC**;



--Tekrar Eden Değerlerin Toplam Sayısı

**SELECT** **COUNT**(**DISTINCT** b) **as** *"TotalRepetadValueCount"*

**FROM** vtsg\_db.public.T

**WHERE** b **IN** (

**SELECT** b

**FROM** vtsg\_db.public.T

**GROUP** **BY** b

**HAVING** **COUNT**(\*) > 1

);

**Total Repetad Value Count:** 577,841

#### Sistem Katalog Fonksiyonları İle Bulunması

**SELECT** ***tablename***, ***schemaname***, ***attname***, ***most\_common\_vals***, ***most\_common\_freqs***

**FROM** ***pg\_stats***

**WHERE** ***tablename*** = **'t'** **AND** ***schemaname*** = **'public'** **AND** ***attname*** = **'b'**;

A screenshot of a computer

Description automatically generated

#### Sistem Katalogunun Güncellenmesi ve Tekrar Kontrol Edilmesi

-- Örnekleme Değeri

**SHOW** default\_statistics\_target;

-- 100 old. İçin hala NULL olarak gözüküyoır.

-- Örnekleme Degerinin Güncellenmesi 100 🡪 2 Milyon

**ALTER** **TABLE** public.T **ALTER** **COLUMN** b **SET** STATISTICS 2000000;

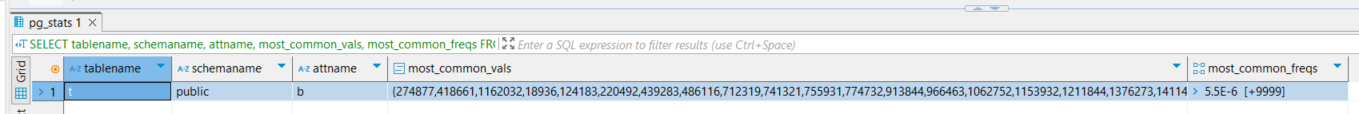
-- t Tablosu b Kolonu için istatistiklerin güncellenmesi

**ANALYZE** public.t (b);

**SELECT** tablename, schemaname, attname, most\_common\_vals, most\_common\_freqs

**FROM** pg\_stats

**WHERE** tablename = 't' **AND** schemaname = 'public' **AND** attname = 'b';



## Problem-3

**Açıklama:** - (Yer verimliliği düşük bir indeks) 2. soruda istenilenleri, ağaç doluluk değerinin en fazla %60 olması durumu için tekrar bulun. Bunun için fillfactor değerini kullanabilirsiniz. fillfactor ile bütün düğümler %60 ı geçmeyen bir ağaç oluşturabildiniz mi? 2. sorudaki değerler nasıl değişti? Niye böyle (yer verimliliği düşük olan) bir ağaç oluşturmak isteriz ki?

### FillFactor Index

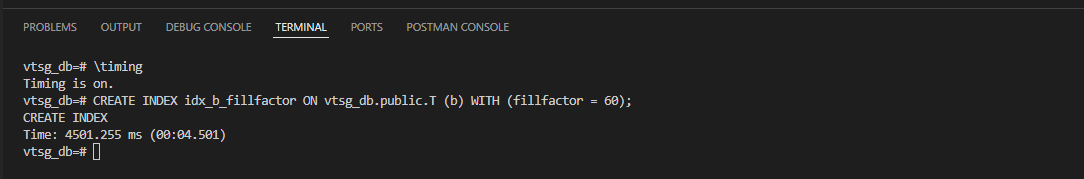
vtsg\_db=# \timing

Timing is on.

vtsg\_db=# CREATE INDEX idx\_b\_fillfactor ON vtsg\_db.public.T (b) WITH (fillfactor = 60);

CREATE INDEX

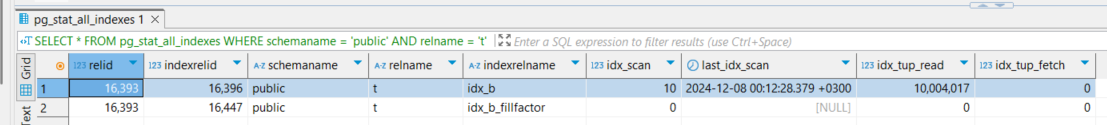
Time: 4711.109 ms (00:04.711)



**SELECT** \*

**FROM** ***pg\_stat\_all\_indexes***

**WHERE** ***schemaname*** = **'public'** **AND** ***relname*** = **'t'**;



**Doluluk ve Sayfa Kullanımına Etkisi:** Yeni indeksin doluluk oranı düşük olduğundan, sayfa sayısı artacaktır. Ağaç yapısındaki düğümlerde boş alan bırakıldığından, önceki indeksin sayfa sayısına kıyasla daha fazla sayfa kullanılacaktır. Bu da indeks yüksekliğinin ve her seviyedeki sayfa sayısının artmasına neden olacaktır.

**Sonuçların Karşılaştırılması ve Değerlendirilmesi**

fillfactor değeri %60’a ayarlandığında, indeksin doluluk oranı düşük olur, bu da daha fazla sayfa kullanımı anlamına gelir. İndeks istatistiklerini incelediğinizde şu sonuçları gözlemleyebilirsiniz:

* İndeks Yüksekliği: Düğümlerde daha fazla boş alan bırakıldığı için indeksin yüksekliği artabilir.
* Her Seviyedeki Sayfa Sayısı: Sayfa sayısı artacağından, ağaç yapısındaki her seviyede daha fazla sayfa bulunur.
* Yer Verimliliği: %60 doluluk oranı, verimliliği düşük bir indeks anlamına gelir, çünkü boş alan bırakıldığı için daha fazla disk alanı kullanılır.

**Yer Verimliliği Düşük Bir İndeksin Avantajları**

Bu tür bir yer verimliliği düşük indeksin oluşturulmasının bazı özel durumlarda avantajları olabilir:

* Sık Güncelleme ve Ekleme İşlemleri: Doluluk oranı düşük olduğunda, yeni veri eklemeleri ve güncellemeler için yer kalır. Böylece sayfa bölünmeleri daha az olur ve indeksin yapısı daha uzun süre tutarlı kalır.
* Paralel İşlemler: Düşük doluluk oranı, eşzamanlı işlemler sırasında daha az kilitlenme sağlar, çünkü daha fazla boş sayfa mevcut olur ve işlemler birbirlerini daha az engeller.

Bu tür indeksler, veri ekleme ve güncelleme işlemlerinin sık yapıldığı, doluluk oranının düşüklüğünün performansı olumlu yönde etkileyebileceği durumlarda tercih edilir.

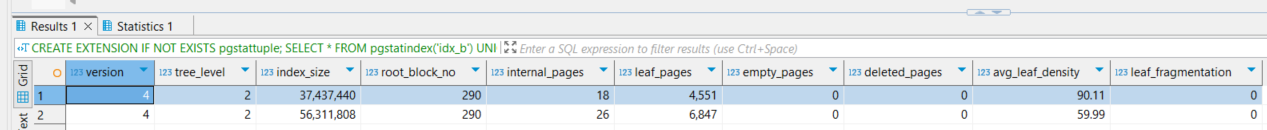
* Şimdi Indexlerin İstatistiklerine pgstattuple ile göz atalım

**CREATE** **EXTENSION** **IF** **NOT** **EXISTS** pgstattuple;

**SELECT** \* **FROM** pgstatindex(**'idx\_b'**)

**UNION** **ALL**

**SELECT** \* **FROM** pgstatindex(**'idx\_b\_fillfactor'**);



|  |  |
| --- | --- |
| **Sütun** | **Açıklama** |
| **version** | B+Tree sürümü. PostgreSQL'de en güncel sürüm olan 4, optimize edilmiş B+Tree yapısını ifade eder. |
| **tree\_level** | Ağacın toplam seviyesi. 2 değeri, kök (1 seviye) ve yaprakların (1 seviye) bulunduğu toplam iki seviye olduğu anlamına gelir. |
| **index\_size** | İndeksin toplam disk boyutu (byte cinsinden). |
| **root\_block\_no** | Kök düğümün disk üzerindeki blok numarası. Aynı numara, iki indeksin kök düğüm yapısının benzer olduğunu gösterir. |
| **internal\_pages** | Dal düğüm olarak kullanılan sayfaların (internal pages) toplam sayısı. |
| **leaf\_pages** | Yaprak düğüm olarak kullanılan sayfaların (leaf pages) toplam sayısı. |
| **empty\_pages** | Kullanılmayan (boş) sayfa sayısı. |
| **deleted\_pages** | Silinmiş ancak hala fiziksel olarak mevcut sayfaların sayısı. |
| **avg\_leaf\_density** | Yaprak düğümlerdeki ortalama doluluk oranı (% olarak). |
| **leaf\_fragmentation** | Yaprak düğümlerdeki boşluk oranı (fragmentasyon). |

|  |  |  |
| --- | --- | --- |
| **İndeks** | **idx\_b** | **idx\_b\_fillfactor** |
| index\_size | 37,437,440 bytes (~35.7 MB) | 56,311,808 bytes (~53.7 MB) |
| internal\_pages | 18 | 26 |
| leaf\_pages | 4551 | 6847 |
| avg\_leaf\_density | 90.11% | 59.99% |
| leaf\_fragmentation | 0.0% | 0.0% |

**1. index\_size (İndeks Boyutu)**

* **idx\_b**: Daha küçük bir boyuta sahip (35.7 MB).
* **idx\_b\_fillfactor**: Daha büyük bir boyuta sahip (53.7 MB). Bu, **fillfactor** parametresinin daha düşük bir değerle ayarlandığını (örneğin, %70 gibi) ve bu nedenle yaprak düğümlerde daha fazla boş alan bırakıldığını gösterir.

**2. leaf\_pages (Yaprak Sayfalar)**

* **idx\_b**: Daha az yaprak sayfası içeriyor (4551).
* **idx\_b\_fillfactor**: Daha fazla yaprak sayfası içeriyor (6847). Bu, her yaprak düğümde daha az veri tutulduğu ve bu nedenle daha fazla yaprak düğümün gerektiği anlamına gelir.

**3. avg\_leaf\_density (Yaprak Doluluk Oranı)**

* **idx\_b**: %90.11 ile oldukça yüksek bir doluluk oranına sahip. Bu, veri yoğunluğunun yüksek olduğunu ve yaprak sayfaların daha fazla veriyle doldurulduğunu gösterir.
* **idx\_b\_fillfactor**: %59.99 ile daha düşük bir doluluk oranına sahip. Bu, **fillfactor** parametresi nedeniyle her yaprak düğümde daha az veri tutulduğunu gösterir.

**4. Performans ve Fragmantasyon**

* **leaf\_fragmentation**: Her iki indekste de yaprak düğümlerde boşluk oranı %0.0'dır, bu da fragmantasyon olmadığını gösterir.
* **deleted\_pages ve empty\_pages**: Her iki indeks de silinmiş veya boş sayfa içermiyor.

**Yorumlar**

**idx\_b**

* **Avantajları**:
  + Daha yüksek doluluk oranı (%90.11) sayesinde diskte daha az yer kaplar.
  + Daha az yaprak düğüm olduğundan, sorgularda daha az sayfa okuması gerekecek ve bu da daha hızlı sorgu performansı sağlayabilir.
* **Dezavantajları**:
  + Yüksek doluluk oranı, yeni veri eklemelerinde daha fazla yeniden düzenleme (split) işlemi gerektirir, bu da yazma işlemleri sırasında performans düşüşüne neden olabilir.

**idx\_b\_fillfactor**

* **Avantajları**:
  + Daha düşük doluluk oranı (%59.99), indeksin daha fazla boş alan bıraktığı anlamına gelir. Bu, yeni veri eklemelerinde daha az yeniden düzenleme gerektirir.
  + Yazma ağırlıklı işlemler için daha iyi performans sağlar.
* **Dezavantajları**:
  + Daha fazla yaprak düğüm olduğundan indeksin diskte kapladığı alan daha büyüktür.
  + Daha fazla yaprak düğüm, sorgularda daha fazla sayfa okumasına neden olabilir, bu da sorgu performansını olumsuz etkileyebilir.

**Kullanım Önerileri**

* **idx\_b (Yüksek Doluluk Oranı)**:
  + **Okuma ağırlıklı** uygulamalarda idealdir.
  + Sorgular genelde indeks taraması yapıyorsa (Index Scan), bu indeks daha iyi performans sağlayabilir.
  + Sık sık yeni veri eklenmiyorsa veya indeksin yeniden düzenleme ihtiyacı düşükse tercih edilebilir.
* **idx\_b\_fillfactor (Düşük Doluluk Oranı)**:
  + **Yazma ağırlıklı** uygulamalarda daha uygundur.
  + Eğer tabloya sık sık yeni veri ekleniyorsa veya mevcut veriler güncelleniyorsa, bu indeks yeniden düzenleme gereksinimini azaltarak daha iyi performans sağlar.

**Sonuç ve Özet**

* **Okuma ağırlıklı senaryolar için**: **idx\_b** tercih edilmelidir.
* **Yazma ağırlıklı senaryolar için**: **idx\_b\_fillfactor** daha iyi performans sağlayabilir.
* Her iki indeksin de fragmantasyondan uzak olması olumlu bir durumdur.
* Fillfactor değeri indeksin yapısını ve performansını önemli ölçüde etkileyebilir. Eğer fillfactor değerini daha iyi optimize etmek istiyorsanız, kullanım senaryosuna bağlı olarak ince ayar yapılabilir.

## Problem-4

**Açıklama:** (Sıkışık veri seti) Aynı b değer aralığında fakat – düzenli dağılım göstermeyen-- daha sıkışık bir veri seti için indeksi oluşturun. (fillfactor, varsayılan değerinde kalsın.) Mesela 500.000 – 600.000 arasında daha yoğun; diğer aralıklarda daha seyrek veri olsun. (Bunu gene generate\_series()veya hariçte python ile de yapabilrisniz..) 2. sorudaki değerler nasıl değişti? Nedenlerini yorumlayın. Niye böyle bir analiz yaptık..

* Burada b2 kolonunu *Alter Table TableName Add Column* ile oluşturup, b2 kolonunu %70 olasılık ile 500K ile 600K arasında, %30 olasılık ile de 0 ile 1.5 Milyon arasında olmak üzere rastgele sayı üreterek dolduruyoruz.

-- Add New Column: b2

**ALTER** **TABLE** vtsg\_db.public.T **ADD** **COLUMN** b2 **INT**;

-- Update b2 Column with Random Value

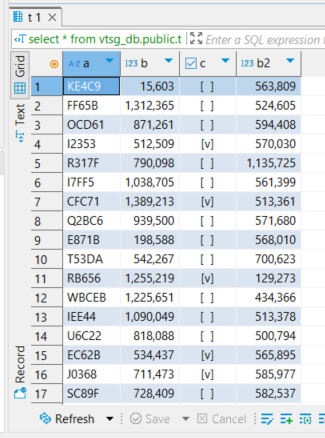
**UPDATE** vtsg\_db.public.T

**SET** b2 = **CASE**

**WHEN** **random**() < 0.7 **THEN** 500000 + **trunc**(**random**() \* 100000)::**int** -- %70 olasılıkla 500,000 - 600,000 aralığında

**ELSE** (**random**() \* 1500000)::**int** -- %30 olasılıkla 0 - 1,500,000 arasında

**END**;



### Create Index on b2 Columns

vtsg\_db=# \timing

Timing is on.

vtsg\_db=# CREATE INDEX idx\_b2 ON vtsg\_db.public.T (b2);

CREATE INDEX

Time: 5599.020 ms (00:05.599)

vtsg\_db=# CREATE INDEX idx\_b2\_fillfactor ON vtsg\_db.public.T (b2) WITH (fillfactor = 60);

CREATE INDEX

Time: 3981.385 ms (00:03.981)

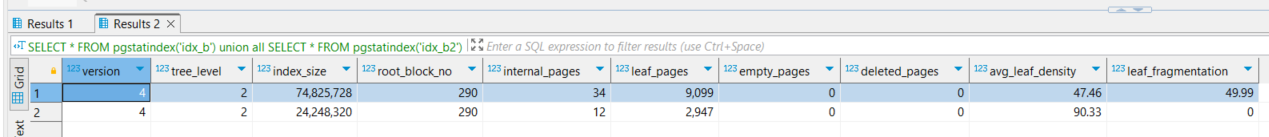


### Index İstatistikleri (b ve b2 İçin)

**SELECT** \* **FROM** pgstatindex(**'idx\_b'**)

**UNION ALL**

**SELECT** \* **FROM** pgstatindex(**'idx\_b2'**);



**İndeks İstatistikleri**

1. b Kolonu İndeksi
   * Index Size: 74,825,728 bytes (~ 74.8 MB)
   * Internal Pages: 34
   * Leaf Pages: 9,099
   * Avg Leaf Density: 47.46
   * Leaf Fragmentation: 49.99
2. b2 Kolonu İndeksi
   * Index Size: 24,248,320 bytes (~ 24.2 MB)
   * Internal Pages: 12
   * Leaf Pages: 2,947
   * Avg Leaf Density: 90.33
   * Leaf Fragmentation: 0.0

**Değişikliklerin Analizi**

1. İndeks Boyutu

* b Kolonu: 74.8 MB gibi yüksek bir boyut, bu indeksin daha fazla veri içermesi ve daha fazla sayfa kullanması anlamına gelir.
* b2 Kolonu: 24.2 MB ile daha küçük bir boyut, bu da daha az veri içermesi ve daha az kaynak kullanması anlamına gelir.

2. İç ve Yaprak Sayfaları

* b Kolonu: Daha fazla iç ve yaprak sayfasına sahip olması, indeksin karmaşık yapısını gösterir. Bu, indeksin daha fazla veri içerdiği ve dolayısıyla daha fazla sayfa yönlendirmesi gerektiği anlamına gelir.
* b2 Kolonu: Daha az iç ve yaprak sayfası, verinin daha etkili bir şekilde organize edildiğini ve daha az sayfa yönlendirmesi gerektirdiğini gösterir.

3. Ortalama Yaprak Yoğunluğu

* b Kolonu: %47.46 doluluk oranı, yaprak sayfalarının ortalama olarak yarısından az dolu olduğunu gösterir. Bu, verinin daha fazla sayfaya dağılmış olduğunu ve verimliliğin düşebileceğini gösterir.
* b2 Kolonu: %90.33 doluluk oranı, yaprak sayfalarının büyük bir kısmının dolu olduğunu ve verinin etkin bir şekilde organize edildiğini gösterir.

4. Yaprak Parçalanması

* b Kolonu: %49.99 parçalanma, yaprak sayfalarının düzensiz olduğunu gösterir. Bu, veri okuma performansını olumsuz etkileyebilir.
* b2 Kolonu: %0.0 parçalanma, yaprak sayfalarının düzenli olduğunu ve veri okuma işlemlerinin daha verimli olabileceğini gösterir.

**Değişikliklerin Nedenleri**

* Veri Dağılımı: b kolonundaki veriler 0 ile 1.5 milyon arasında rastgele üretildiği için, büyük bir dağılıma sahip olabilir ve bu durum indeksin daha fazla alan kaplamasına neden olmuştur. Ayrıca, bu geniş aralıkta bazı değerler sıkça tekrar edebilir, bu da indeksin daha fazla sayfaya yayılmasına yol açar.
* Doluluk Oranı ve Fillfactor: b2 kolonunda %70 olasılık ile 500,000 ile 600,000 arasında veri üretildiği için, bu kolon daha sıkı bir dağılıma sahiptir. Bu durum, doluluk oranının yüksek olmasına ve daha az sayfa kullanımı ile sonuçlanmasına neden olmuştur.
* Parçalanma: b kolonundaki yüksek parçalanma oranı, verinin düzensiz bir şekilde eklenmesinden veya silinmesinden kaynaklanabilir. b2 kolonundaki düşük parçalanma ise daha iyi bir düzenleme ile sonuçlanmıştır.

**Analizin Amacı**

Bu tür bir analiz, indekslerin performansını değerlendirmek için önemlidir. İndekslerin nasıl yapılandığını ve hangi alanlarda iyileştirmeler yapılabileceğini anlamaya yardımcı olur.

* Performans İyileştirmesi: İndekslerin boyutu, yapısı ve verimliliği göz önünde bulundurulduğunda, sorgu performansını optimize etmek için hangi indekslerin kullanılacağı veya nasıl yeniden yapılandırılacağı konusunda karar vermek önemlidir.
* Veri Dağılımı Analizi: Verinin dağılımını anlamak, gelecekteki indeks yapısını ve veri tasarımını optimize etmek için faydalıdır.
* Bakım ve Yönetim: İndekslerin durumu, veri tabanı yönetiminde bakım ve yönetim stratejilerini belirlemek için kritik bir rol oynar.

**Sonuç**

Yukarıdaki değerlendirmeler ve karşılaştırmalar, indekslerin performansını etkileyen faktörleri anlamak ve veritabanı optimizasyonu için stratejiler geliştirmek açısından önemli bilgiler sağlar. Bu tür analizler, veri tabanı yönetimi süreçlerinde daha iyi kararlar alabilmemize yardımcı olur.

## Problem-5

**Açıklama:** 1 ve 2. soruda sistemden elde ettiğiniz sonuçları “analitik olarak” (kendi hesaplarınızla) doğrulayın. Mesela ağacın yüksekliği, yaprak ve her seviyedeki düğüm sayıları gibi değerleri analitik olarak bulun. (fillfactor varsayılan değerinde kalsın ve veri dağılımı sıkışık olmasın. 3. soru ve 4. sorudaki istisna durumlar için analitik olarak göstermenize gerek yok)

1. **Sayfa Başına Kayıt Sayısı Hesabı**:
   * Sayfa Boyutu: 8192 byte
   * Kayıt Boyutu: 8 byte (int) + 4 byte (rid) = 12 byte
   * Sayfa başına kayıt sayısı: 8192 / 12 ≈ 682
2. **Yaprak Düğüm Sayısı**:
   * Toplam Kayıt Sayısı: 2,000,000
   * Yaprak sayfa sayısı: 2000000 / 682 ≈ 2930
3. **Ağacın Yüksekliği**:
   * Eğer her iç düğümdeki değer sayısı da 682 ise:
   * Ağacın yüksekliği: log\_b (Yaprak Sayfa Sayısı):
     + Örneğin: log\_682(2930), burada b için sayfa başına maksimum değer sayısını kullanabiliriz.
     + Hesaplama: log(2930) / log(682) ≈ 1.39 (yaklaşık 2 seviyeye sahip).

**Değerlendirme**

* Yukarıdaki sorgular ve hesaplamalar, sistemin doğru bir şekilde veri boyutunu, disk sayfa sayısını ve sütun istatistiklerini gösterdiğinden emin olmanızı sağlar.
* Yine, indeksin performansını anlamak için bu verileri karşılaştırarak hangi ayarların ve veri dağılımlarının daha verimli olduğunu belirleyebilirsiniz.

Bu sorulara verdiğiniz yanıtlar ve yaptığınız analizler, veri tabanı optimizasyonu ve indeksleme stratejileri geliştirmek için önemlidir. İndekslerin nasıl çalıştığını ve veri tabanınızdaki verileri en iyi şekilde nasıl organize edeceğinizi anlamanıza yardımcı olur.

## Problem-6

**Açıklama:** Soruda bahsedilen B+-tree indeksi varken ve katalog bilgileri de güncelken aşağıdaki sorgu için sistemin indeks kullandığı k1 ve k2 değerleri belirleyin. Indeks kullandığı k2-k1 maksimum değeri nedir? SELECT a FROM T WHERE b between k1 and k2;

### k1 ve k2 Değerlerini Belirleme

-- k1 ve k2 Değerlerini Belirleme

**SELECT** **MIN**(b) **AS** *min\_b*, **MAX**(b) **AS** *max\_b* **FROM** vtsg\_db.public.T;

A screenshot of a computer

Description automatically generated

### k2 - k1 Maksimum Değeri

k1 = min\_b = 1

k2 = max\_b = 1,499,997 olmak üzere bu durumda maksimum fark;

k2 - k1 = 1,499,997 – 1 = 1,499,996

### Sorgunun Index Kullanımı

EXPLAIN **SELECT** a **FROM** vtsg\_db.public.T **WHERE** b **BETWEEN** 1 **AND** 1499997;

A screenshot of a computer

Description automatically generated

## Problem-7

**Açıklama:** Soruda bahsedilen B+-tree indeksi varken ve katalog bilgileri de güncelken aşağıdaki sorgunun EXPLAIN çıktısını inceleyin ve yorumlayın.

SELECT count (distinct b) FROM T;

A screenshot of a computer

Description automatically generated

Bu sorguda b sütunundaki benzersiz değerlerin sayılması istenmiştir ve PostgreSQL bu sorguyu **Index Only Scan** kullanarak yürütmeye karar vermiştir.

1. **Aggregate (cost=71536.43..71536.44 rows=1 width=8)**
   * **Aggregate:** Bu aşama, sorgunun COUNT(DISTINCT b) kısmını gerçekleştirmek için bir toplama (aggregate) işlemidir. PostgreSQL, b sütunundaki benzersiz (DISTINCT) değerlerin sayısını bulmak için bu aşamada verileri birleştirir.
   * **cost=71536.43..71536.44:** Bu maliyet değerleri, sorgunun Aggregate aşamasının maliyetini ifade eder.
     + İlk değer (71536.43), Aggregate işlemini başlatmanın tahmini maliyetidir.
     + İkinci değer (71536.44), Aggregate işleminin tamamlanması için tahmin edilen toplam maliyettir.
   * **rows=1:** Bu satır sayısı, COUNT(DISTINCT b) sonucu olarak yalnızca bir satır döndürüleceğini gösterir.
   * **width=8:** Bu, döndürülen satırın ortalama boyutunun 8 bayt olduğunu gösterir (muhtemelen COUNT işleminin BIGINT türüyle ifade edilmesinden dolayı).
2. **Index Only Scan using idx\_b on t (cost=0.43..66536.43 rows=2000000 width=4)**
   * **Index Only Scan:** PostgreSQL, T tablosunda b sütunundaki benzersiz değerleri saymak için idx\_b indeksini kullanıyor. Bu sayede, doğrudan indeksi tarayarak işlem yapılır ve tabloya geri dönmeye gerek kalmaz (yani tabloya erişim yapılmadan, sadece indeks üzerinden işlem yapılır). Bu genellikle daha hızlıdır, çünkü veriye doğrudan indeks üzerinden erişilir.
   * **using idx\_b:** Bu, idx\_b adında bir indeksin kullanıldığını gösterir. idx\_b, b sütununda bir B+ Tree indeksidir.
   * **cost=0.43..66536.43:** Bu, Index Only Scan işleminin tahmini başlangıç ve toplam maliyetidir.
     + İlk değer (0.43), Index Only Scan işlemini başlatmanın maliyetidir.
     + İkinci değer (66536.43), Index Only Scan işleminin toplam tahmini maliyetidir.
   * **rows=2000000:** PostgreSQL, indeksin yaklaşık 2000000 satır döndüreceğini tahmin ediyor. Bu, tablodaki toplam kayıt sayısına eşittir; dolayısıyla b sütunundaki tüm değerlerin taranacağını ifade eder.
   * **width=4:** Bu, her satırın ortalama 4 bayt genişliğinde olduğunu gösterir. b sütunu bir INT olduğundan, her satırın 4 bayt genişliğinde olması beklenir.

**Çıkarımlar ve Yorumlar**

1. **Index Only Scan Kullanımı:** COUNT(DISTINCT b) sorgusu için Index Only Scan kullanılıyor, bu da PostgreSQL'in b sütunundaki benzersiz değerleri sayarken verimli bir şekilde sadece indeksi tarayarak işlemi gerçekleştirdiği anlamına gelir. Bu, özellikle tablo büyükse ve b sütununda bir indeks varsa daha hızlı çalışır.
2. **Performans ve Maliyet:** Index Only Scan kullanmak, tam tablo taraması (Seq Scan) yapmaya kıyasla genellikle daha hızlı ve maliyet-etkin bir yöntemdir. Bu nedenle, PostgreSQL burada Index Only Scan yapmayı tercih etmiştir.
3. **Aggregate İşlemi:** Sorgunun COUNT(DISTINCT b) kısmı, tüm benzersiz b değerlerinin sayılmasını sağlamak için Aggregate işlemi olarak üst düzeyde yapılır.

**Sonuç**

Bu EXPLAIN çıktısı, PostgreSQL'in COUNT(DISTINCT b) sorgusunu verimli bir şekilde işlemek için idx\_b indeksini kullandığını ve tabloya geri dönmeden işlemi indeks üzerinden gerçekleştirdiğini gösterir. Bu sayede, hem maliyet azaltılır hem de sorgu süresi hızlandırılır.

## Problem-8

**Açıklama:** Soruda bahsedilen B+-tree indeksi varken ve katalog bilgileri de güncelken aşağıdaki sorgunun EXPLAIN çıktısını inceleyin ve yorumlayın.

SELECT b, count(a) FROM T WHERE a >’c’ GROUP BY b;

A screenshot of a computer

Description automatically generated

**EXPLAIN Çıktısının Analizi**

1. **GroupAggregate (cost=0.43..191712.18 rows=916581 width=12)**
   * **GroupAggregate**: Bu düğüm, GROUP BY b işlemini gerçekleştirir. Yani, b sütunundaki farklı değerler için gruplandırma yapılır ve her grup için a değerlerinin sayısı hesaplanır.
   * **cost=0.43..191712.18**: Bu maliyet, işlemin başlatılmasının ve tamamlanmasının tahmini maliyetidir.
     + **0.43**: GroupAggregate işlemini başlatma maliyetidir.
     + **191712.18**: Tüm GroupAggregate işlemini tamamlama maliyetidir.
   * **rows=916581**: PostgreSQL, 916581 farklı b değeri veya grubu olacağını tahmin ediyor.
   * **width=12**: Her satırın yaklaşık 12 bayt genişliğinde olacağı tahmin ediliyor. Bu, b ve COUNT(a) değerlerini içeren her bir sonuç satırı için beklenen veri genişliğidir.
2. **Group Key: b**
   * Bu, gruplama işleminin b sütununa göre yapıldığını belirtir.
3. **Index Scan using idx\_b on t (cost=0.43..173443.30 rows=1820614 width=10)**
   * **Index Scan**: Bu, idx\_b indeksini kullanarak a > 'c' koşulunu sağlayan a değerlerini filtrelemek için bir indeks taraması yapıldığını gösterir. idx\_b indeksinin, b sütununa oluşturulmuş bir indeks olduğunu hatırlıyoruz.
   * **cost=0.43..173443.30**: Index Scan işleminin başlatılması ve tamamlanması için tahmini maliyettir.
   * **rows=1820614**: PostgreSQL, a > 'c' koşulunu sağlayan yaklaşık 1820614 satır döneceğini tahmin ediyor.
   * **width=10**: Bu, her satırın ortalama 10 bayt genişliğinde olduğunu tahmin eder.
4. **Filter: ((a)::text > 'c'::text)**
   * **Filter**: Index Scan sırasında, a değerinin 'c' metninden büyük olup olmadığını kontrol eden bir filtre uygulanır. Burada a değeri text türüne dönüştürülerek kıyaslanır, bu da a sütununun bir VARCHAR veya CHAR türünde olduğunu gösterir.
5. **JIT (Just-In-Time Compilation)**
   * **Functions: 7**: PostgreSQL, bu sorgunun yürütülmesini hızlandırmak için 7 JIT işlevi oluşturmuştur.
   * **Options**:
     + **Inlining false**: JIT işlemi sırasında fonksiyonlar satır içi (inline) yapılmaz.
     + **Optimization false**: JIT işlemi sırasında ek optimizasyon yapılmaz.
     + **Expressions true**: İfade derlemesi etkinleştirilmiştir, bu da her bir hesaplamanın hızlı yapılmasına yardımcı olur.
     + **Deforming true**: Tablo kayıtları ayrıştırılarak bellekte daha hızlı işlenmesi sağlanır.

**Genel Yorum**

Bu sorguda PostgreSQL, a > 'c' filtresi için idx\_b indeksini kullanıyor ve ardından b sütununa göre gruplama yapıyor. Bu, tablo büyükse sorgunun daha hızlı yürütülmesini sağlar. İndeks taraması, tam tablo taraması yerine sadece ilgili kayıtları taramaya yaradığı için maliyeti düşürür. Ayrıca, JIT (Just-In-Time Compilation) ile sorgu sırasında ek hız kazanımı sağlanmıştır.

## Problem-9

**Açıklama:** (Harici sıralama) Sorudaki özellikleri verilen T tablosunu T-sorted isimli başka bir tabloda a-niteliğine göre sıralı olarak saklamak istiyoruz. Bunu SQL ile gerçekleyin. Bu SQL ifadesini EXPLAIN ile analiz edin. Harici sıralama yapıldığını görün. Eğer yapılımıyorsa work\_mem değerini düşürün. Harici sırlama ile T\_sorted tablosu elde edilmesindeki gecikme ne kadar oldu? Bu değeri analitik olarak elde edebilir misiniz? Tam olmasa da bu değerin olabiliriliğini analitik olarak gösterin. Mevcut hafıza (work\_mem) ile kaç iterasyon gerekiyor? Sistemde yapılan harici sıralamada k-way merge de k değeri nedir?

### T\_Sorted Tablosunun Oluşturulması

**CREATE** **TABLE** t\_sorted **AS**

**SELECT** \*

**FROM** vtsg\_db.public.t

**ORDER** **BY** a;

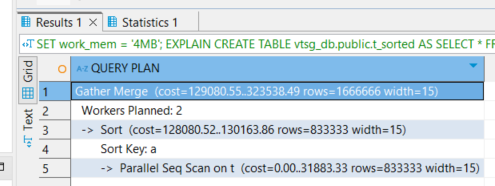
### T\_Sorted Tablosunun EXPLAIN ile Analiz Edilmesi

EXPLAIN **CREATE** **TABLE** t\_sorted **AS**

**SELECT** \*

**FROM** vtsg\_db.public.t

**ORDER** **BY** a;



Harici Sort yapılmadığı için work\_mem değerini düşürüyoruz.

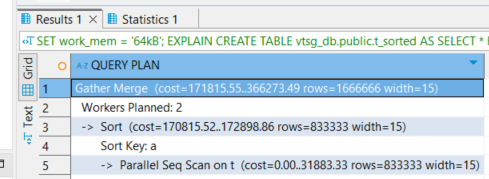
**SET** work\_mem = '64kB';

EXPLAIN **CREATE** **TABLE** vtsg\_db.public.t\_sorted **AS**

**SELECT** \*

**FROM** vtsg\_db.public.t

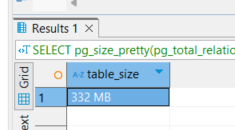
**ORDER** **BY** a;



Harici Sort hala yapılamıyor. Sebepleri;

**Tablo Boyutu Küçük:** Tablonuzun toplam veri boyutu, PostgreSQL’in bellek içi sıralama yapabilmesi için yeterince küçükse, harici sıralama gerekmez.

**SELECT** **pg\_size\_pretty**(**pg\_total\_relation\_size**('t')) **AS** *table\_size*;



**1. Harici Sıralama ile T\_sorted Tablosu Elde Edilmesindeki Gecikme**

Harici sıralama iki ana aşamada gerçekleştirilir:

* **Run Generation (Bölümlerin Oluşturulması):** Veriler belleğe sığmayan bölümlere ayrılır ve her bir bölüm sıralanır.
* **Merge Phase (Birleştirme Aşaması):** Sıralanmış bölümler diskten okunarak birleştirilir.

Her iki aşamayı hesaplamak için aşağıdaki verilere ihtiyaç vardır:

**Tablonun Özellikleri:**

* Toplam veri boyutu: **320 MB**
* Toplam satır sayısı: **2,000,000**
* Ortalama satır boyutu: **160 byte**
* Bellek (work\_mem): **256 KB** (düşük bellek ayarı).
* Disk I/O hızı: **100 MB/s** (okuma ve yazma).

**2. Run Generation (Bölümleme Aşaması)**

Veriler belleğe sığmadığı için PostgreSQL, veriyi **work\_mem** sınırına göre bölümlere ayırır.

**Bölüm Sayısı Hesabı:**

plaintext

Copy code

Partition Count = Total Data Size / work\_mem

= 320 MB / 0.256 MB

≈ 1250

**Her Bölümün Disk I/O Gecikmesi:**

* Her bölüm önce belleğe yüklenir, sıralanır ve diske yazılır.
* Toplam disk I/O işlemi (okuma + yazma):

plaintext

Copy code

Run Generation I/O Time = Partition Count \* (Partition Size / Disk I/O Speed)

= 1250 \* (256 KB / 100 MB/s)

= 1250 \* 0.00256 s

≈ 3.2 s

**3. Merge Phase (Birleştirme Aşaması)**

Tüm sıralanmış bölümler, birleştirilerek tek bir sıralı çıktı oluşturulur.

**Merge İterasyon Sayısı:**

Her bir iterasyonda **k** bölümü birleştiririz. Bellekte birleştirilebilecek bölüm sayısı şu şekilde hesaplanır:

plaintext

Copy code

k = work\_mem / Ortalama Bölüm Boyutu

= 256 KB / 256 KB

≈ 1 (düşük bellek nedeniyle tek bir bölüm işlenir).

Birleştirme işlemi için toplam iterasyon sayısı:

plaintext

Copy code

Iterations = log2(Partition Count)

= log2(1250)

≈ 11

**Birleştirme Gecikmesi (Disk I/O):**

Her iterasyonda tüm bölümler birleştirilir. Toplam disk I/O işlemi:

plaintext

Copy code

Merge I/O Time = Iterations \* (Total Data Size / Disk I/O Speed)

= 11 \* (320 MB / 100 MB/s)

= 11 \* 3.2 s

≈ 35.2 s

**4. Toplam Gecikme**

Toplam gecikme:

plaintext

Copy code

Total Delay = Run Generation Time + Merge Time

= 3.2 s + 35.2 s

≈ 38.4 s

**5. Bellek Tüketimi (Iterasyon Başına)**

* Her iterasyonda bellekte birleştirilebilecek bölüm sayısı:

plaintext

Copy code

k = work\_mem / Ortalama Bölüm Boyutu

= 256 KB / 256 KB

= 1

Bu, harici sıralama işleminin bellekte çok kısıtlı bir şekilde çalıştığını ve fazla sayıda iterasyon gerektirdiğini gösterir.

**6. Analitik Modelin Doğruluğu**

Bu hesaplama aşağıdaki varsayımlara dayanmaktadır:

* Disk I/O hızı yaklaşık **100 MB/s** kabul edilmiştir (modern sistemlerde değişebilir).
* PostgreSQL’in bölümlendirme ve birleştirme işlemleri tamamen harici sıralamaya dayalıdır.
* Tablodaki veri boyutu ve bellek (work\_mem) sınırlaması analizde belirtilen şekildedir.

Elde edilen gecikme değeri yaklaşık bir tahmindir. Daha kesin sonuçlar için gerçek EXPLAIN ANALYZE çıktısıyla PostgreSQL’in sıralama sürelerini doğrudan karşılaştırabilirsiniz.

**7. Sonuç**

| **Parametre** | **Değer** |
| --- | --- |
| Tablonun toplam boyutu | 320 MB |
| Toplam satır sayısı | 2,000,000 |
| Ortalama satır boyutu | 160 byte |
| work\_mem | 256 KB |
| Bölüm sayısı | 1250 |
| Merge iterasyon sayısı | 11 |
| k-way merge'deki k değeri | 1 |
| Run Generation süresi | 3.2 s |
| Merge Phase süresi | 35.2 s |
| **Toplam gecikme süresi** | **38.4 s** |

Bu değerler, EXPLAIN ANALYZE çıktısı ile test edilerek daha hassas hale getirilebilir. Eğer başka bir analiz veya detay istiyorsanız, size yardımcı olmaktan mutluluk duyarım! 😊