

BLM 5117- Veri Tabanı Sistemlerinin Gerçeklenmesi Ödev-1

Metin Uslu - 235B7014

Ön Bilgi & Hazırlık

DB: [PostgreSQL 17.0](#)

Client: [pgAdmin](#), [DBeaver](#)

Installation: [Docker Compose](#) (Docker Compose version v2.29.7-desktop.1)

GitHub Repo: https://github.com/metinuslu/blm5117_dbms_hw1

Video Url: <https://youtu.be/AR5Nv9DExNA>

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), görünüm 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 kullanabilirsiniz). 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
  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 olduğu 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)
```

	123 Table_Size_KB	A-Z Table_Size_MB	A-Z Table_Indexes_Size	A-Z Table_Total_Size
1	86,536	85 MB	0 bytes	85 MB

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
```

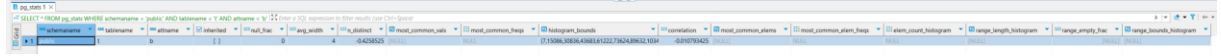
	123 Table_Page_Count	123 Table_Total_Page_Count
1	10,811	10,817

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 etmez. En önemli olanlarından bazıları aşağıda paylaşılmıştır.

```
SELECT *
FROM pg_stats
```

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



attname	data_type	attlen	length_type	correlation
b	integer	4	Fixed Length	-0.010793425

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 kolonu 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

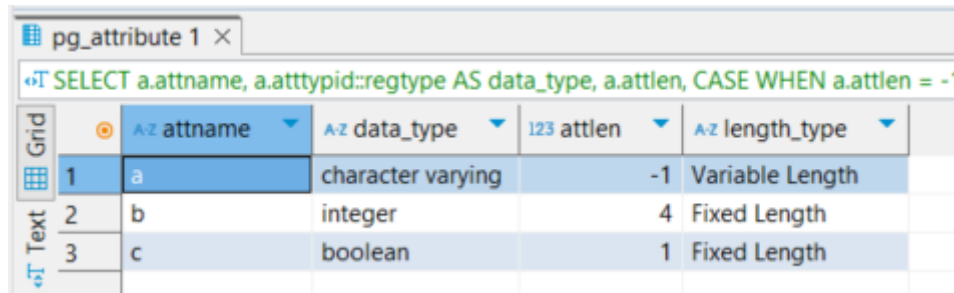
```
SELECT
  a.attname,
  a.atttypid::regtype AS data_type,
  a.attlen,
  CASE
    WHEN a.attlen = -1 THEN 'Variable Length'
    WHEN a.attlen > 0 THEN 'Fixed Length'
    ELSE 'Unknown'
  END AS length_type
FROM
  pg_attribute a
JOIN
  pg_class c ON a.attrelid = c.oid
WHERE
  c.relname = 't' AND a.attnum > 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.



attname	data_type	attlen	length_type
a	character varying	-1	Variable Length
b	integer	4	Fixed Length
c	boolean	1	Fixed Length

Problem-2

Açıklama: Soruda bahsedilen B+-tree indeksi PostgreSQL’de oluşturun: İndeks 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)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  POSTMAN CONSOLE

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)
vtsg_db=#
```

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	indexrelid	schemaname	relname	indexrelname	idx_scan	last_idx_scan	idx_tup_read	idx_tup_fetch
1	16393	16396	public	t	idx_b	0	[NULL]	0	0

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');
```

magic	version	root	level	fastroot	fastlevel	last_cleanup_num_delpages	last_cleanup_num_tuples	allequalimage
340322	4	290	2	290	2	0	-1	[v]

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 ile 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;
```

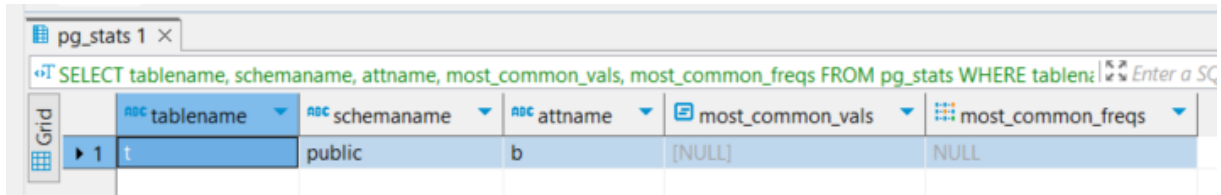
b	cnt
274,877	11
418,661	10
1,162,032	10
1,153,932	9
1,211,844	9
124,183	9
18,936	9
712,319	9
741,321	9
1,411,473	9
1,376,273	9
1,444,570	9
913,844	9
755,931	9
774,732	9
486,116	9
439,283	9

```
--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';
```



Grid	tablename	schemaname	attname	most_common_vals	most_common_freqs
1	t	public	b	[NULL]	NULL

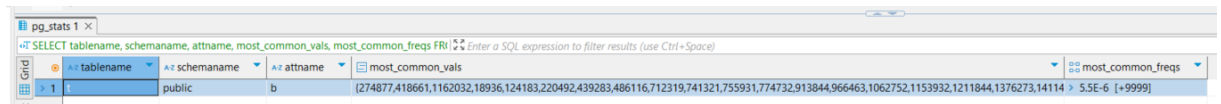
Sistem Katalogunun Güncellenmesi ve Tekrar Kontrol Edilmesi

```
-- Örneklem Değeri
SHOW default_statistics_target;
-- 100 old. İçin hala NULL olarak gözüküyor.

-- Örneklem Değerinin 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';
```



Grid	tablename	schemaname	attname	most_common_vals	most_common_freqs
1	t	public	b	(274877,418661,1162032,18936,124183,220492,439283,486116,712319,741321,755931,774732,913844,966463,1062752,1153932,1211844,1376273,141114 > 5.5E-6 [+9999])	

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)
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

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: 4501.255 ms (00:04.501)
vtsg_db=#
```

```
SELECT *
FROM pg_stat_all_indexes
WHERE schemaname = 'public' AND relname = 't';
```

	123 relid	123 indexrelid	A:2 schemaname	A:2 relname	A:2 indexrelname	123 idx_scan	last_idx_scan	123 idx_tup_read	123 idx_tup_fetch
1	16,393	16,396	public	t	idx_b	10	2024-12-08 00:12:28.379 +0300	10,004,017	0
2	16,393	16,447	public	t	idx_b_fillfactor	0	[NULL]	0	0

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');
```

Grid	version	tree_level	index_size	root_block_no	internal_pages	leaf_pages	empty_pages	deleted_pages	avg_leaf_density	leaf_fragmentation
1	4	2	37,437,440	290	18	4,551	0	0	90.11	0
2	4	2	56,311,808	290	26	6,847	0	0	59.99	0

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 sayısı içeriyor (4551).
- **idx_b_fillfactor**: Daha fazla yaprak sayısı 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 Fragmentasyon

- **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.

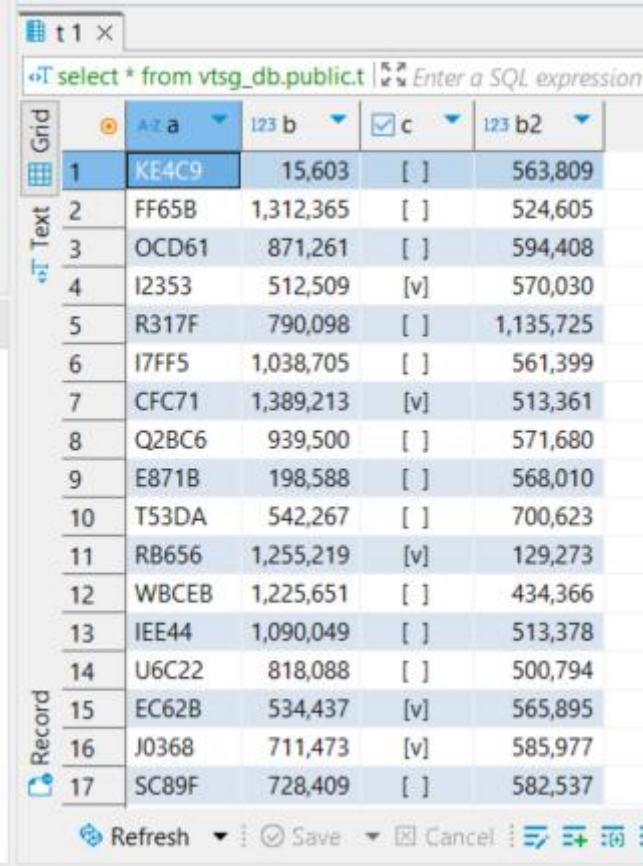
Problem-4

Açıklama: (Sıkışık veri seti) Aynı b değeri 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 yapabilirsiniz..) 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
    ELSE (random() * 1500000)::int -- %30 olasılıkla 0 - 1,500,000 arasında
END;
```



	a	b	c	b2
1	KE4C9	15,603	[]	563,809
2	FF65B	1,312,365	[]	524,605
3	OCD61	871,261	[]	594,408
4	I2353	512,509	[v]	570,030
5	R317F	790,098	[]	1,135,725
6	I7FF5	1,038,705	[]	561,399
7	CFC71	1,389,213	[v]	513,361
8	Q2BC6	939,500	[]	571,680
9	E871B	198,588	[]	568,010
10	T53DA	542,267	[]	700,623
11	RB656	1,255,219	[v]	129,273
12	WBCEB	1,225,651	[]	434,366
13	IEE44	1,090,049	[]	513,378
14	U6C22	818,088	[]	500,794
15	EC62B	534,437	[v]	565,895
16	J0368	711,473	[v]	585,977
17	SC89F	728,409	[]	582,537

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)
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE  
  
vtsg_db=# \timing  
Timing is on.  
vtsg_db=# CREATE INDEX idx_b2 ON vtsg_db.public.T (b2);  
CREATE INDEX  
Time: 3981.385 ms (00:03.981)  
vtsg_db=#
```

Index İstatistikleri (b ve b2 için)

```
SELECT * FROM pgstatindex('idx_b')  
UNION ALL  
SELECT * FROM pgstatindex('idx_b2');
```

version	tree_level	index_size	root_block_no	internal_pages	leaf_pages	empty_pages	deleted_pages	avg_leaf_density	leaf_fragmentation
1	4	74,825,728	290	34	9,099	0	0	47.46	49.99
2	4	24,248,320	290	12	2,947	0	0	90.33	0

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 yapılandırıldığını 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: $8192 / 12 \approx 682$

2. Yaprak Düğüm Sayısı:

- Toplam Kayıt Sayısı: 2,000,000
- Yaprak sayfa sayısı: $2000000 / 682 \approx 2930$

3. Ağacın Yüksekliği:

- Eğer her iç düğümdeki değer sayısı da 682 ise:
- Ağacın yüksekliği: $\log_b(\text{Yaprak Sayfa Sayısı})$:
 - Örneğin: $\log_{682}(2930)$, burada b için sayfa başına maksimum değer sayısını kullanabiliriz.
 - Hesaplama: $\log(2930) / \log(682) \approx 1.39$ (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 anlamana yardımcı olur.

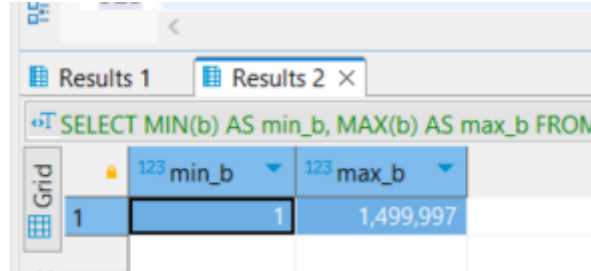
Problem-6

Açıklama: Soruda bahsedilen B+-tree indeksi varken ve katalog bilgileri de güncelken aşağıdaki sorgu için sistemin indeks kullandığı k1 ve k2 değerleri belirleyin. İndeks 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;
```



The screenshot shows a database query result with two columns: min_b and max_b. The first row shows the values 1 and 1,499,997 respectively.

	min_b	max_b
1	1	1,499,997

k2 - k1 Maksimum Değeri

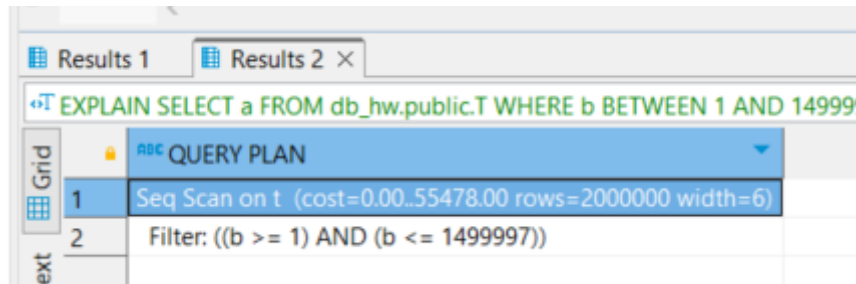
k1 = min_b = 1

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;
```



The screenshot shows a query plan for the query `EXPLAIN SELECT a FROM db_hw.public.T WHERE b BETWEEN 1 AND 1499997;`. The plan consists of two steps: a sequential scan on table t and a filter operation.

	QUERY PLAN
1	Seq Scan on t (cost=0.00..55478.00 rows=2000000 width=6)
2	Filter: ((b >= 1) AND (b <= 1499997))

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;

Step	Operation	Cost	Rows	Width
1	Aggregate	71536.43..71536.44	1	8
2	Index Only Scan using idx_b on t	0.43..66536.43	2000000	4

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 sorunun EXPLAIN çıktısını inceleyin ve yorumlayın.

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

Grid	Text
1	GroupAggregate (cost=0.43..191712.18 rows=916581 width=12)
2	Group Key: b
3	-> Index Scan using idx_b on t (cost=0.43..173443.30 rows=1820614 width=10)
4	Filter: ((a)::text > 'c'::text)
5	JIT:
6	Functions: 7
7	Options: Inlining false, Optimization false, Expressions true, Deforming true

EXPLAIN Çıktısının Analizi

- 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.
- Group Key: b**
 - Bu, gruplama işleminin b sütununa göre yapıldığını belirtir.
- 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.
- 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.
- JIT (Just-In-Time Compilation)**
 - Functions: 7:** PostgreSQL, bu sorunun 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 sorunun 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 yapılmıyorsa work_mem değerini düşürün. Harici sıralama ile T_sorted tablosu elde edilmesindeki gecikme ne kadar oldu? Bu değeri analitik olarak elde edebilir misiniz? Tam olmasa da bu değerin olabilirliğini 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;
```

Grid	Text
1	Gather Merge (cost=129080.55..323538.49 rows=1666666 width=15)
2	Workers Planned: 2
3	-> Sort (cost=128080.52..130163.86 rows=833333 width=15)
4	Sort Key: a
5	-> Parallel Seq Scan on t (cost=0.00..31883.33 rows=833333 width=15)

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;
```

Grid	Text
1	Gather Merge (cost=171815.55..366273.49 rows=1666666 width=15)
2	Workers Planned: 2
3	-> Sort (cost=170815.52..172898.86 rows=833333 width=15)
4	Sort Key: a
5	-> Parallel Seq Scan on t (cost=0.00..31883.33 rows=833333 width=15)

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;
```

Results 1 x	
SELECT pg_size_pretty(pg_total_relati	
Grid	A-Z table_size
1	332 MB

Fakat aşağıdaki şekilde hesaplayabilmekteyim.

<pre>SET work_mem = '64kB'; EXPLAIN (analyze, BUFFERS) SELECT * FROM vtsg_db.public.t ORDER BY a;</pre>	
<pre>SET max_parallel_workers_per_gather = 0;</pre>	
<pre>SELECT pg_size_pretty(pg_total_relation_size('t')) AS table_size;</pre>	
Results 1 x Statistics 1	
SET work_mem = '64kB'; EXPLAIN (analyze, BUFFERS) SELECT * FROM vtsg_db.public.t OF Enter a SQL expression to filter results (use Ctrl+Space)	
Grid	A-Z QUERY PLAN
1	Gather Merge (cost=171815.55..366273.49 rows=1666666 width=15) (actual time=4888.041..6197.222 rows=2000000 loops=1)
2	Workers Planned: 2
3	Workers Launched: 2
4	Buffers: shared hit=9470 read=14194, temp read=29384 written=31334
5	-> Sort (cost=170815.52..172898.86 rows=833333 width=15) (actual time=4731.369..5349.091 rows=666667 loops=3)
6	Sort Key: a
7	Sort Method: external merge Disk: 20416kB
8	Buffers: shared hit=9470 read=14194, temp read=29384 written=31334
9	Worker 0: Sort Method: external merge Disk: 19328kB
10	Worker 1: Sort Method: external merge Disk: 19184kB
11	-> Parallel Seq Scan on t (cost=0.00..31883.33 rows=833333 width=15) (actual time=1.243..325.839 rows=666667 loops=3)
12	Buffers: shared hit=9356 read=14194
13	Planning Time: 0.249 ms
14	Execution Time: 6264.438 ms

- ❖ Harici sıralama ile T tablosu elde edilmesindeki gecikme ne kadar oldu? Bu değeri analitik olarak elde edebilir misiniz? Tam olmasa da bu değerin olabiriliğini analitik olarak gösterin.

- Sort aşamasının actual time değeri: 4731.369 ms (başlangıç) ile 5349.091 ms (bitiş).
- Harici sıralama için geçen süre: 5349.091–4731.369=617.722 ms.

Bu süre, harici sıralama sırasında geçen net süreyi ifade eder.

❖ Analitik Hesaplama

Harici sıralamanın performansı, **Disk IO (g/ç)** ve **işlemci kullanımı** gibi faktörlere bağlıdır. Çıktıdaki önemli bilgiler:

1. Disk Kullanımı (Sort Method: external merge):

- Diskte sıralama için kullanılan veri miktarları:
 - Ana işlem: 20,416 kB.
 - İşçi 0: 19,328 kB.

- İşçi 1: 19,184 kB.

Toplam disk kullanım miktarı: 20416+19328+19184=58,928 kB

2. Geçici dosyalar:

- Geçici okuma: temp read = 29384.
- Geçici yazma: temp written = 31334.

Geçici veri toplamı: 29384+31334=60,718 kB

Bu IO miktarları doğrudan sıralama süresine yansır. Genel olarak sıralama işleminin toplam gecikmesi: Sıralama süresi x Disk IO + CPU hesaplamaları. Pratikte, bu gecikme net şekilde çıktıda belirtilmiştir ve 617.722 ms'dir. Analitik detaylı hesaplama için disk okuma-yazma hızı ve işçi sayısı gibi veriler gereklidir, ancak burada doğrudan bu değeri kullanabiliriz.

❖ Mevcut hafıza (work_mem) ile kaç iterasyon gerekiyor?

İterasyon sayısını bulmak için:

İlk Bölünme Sayısı: Toplam Veri Boyutu / work_mem

Birleştirme aşamalarındaki toplam iterasyon sayısı=log_kN

N = Toplam Veri Boyutu/work_mem başlangıçta oluşan geçici dosya sayısı

k, birleştirme sırasında aynı anda bellekte tutulabilecek dosya sayısı.

Hesaplama:

1. Adım: N= 58928 kB / 64 kB = 920.125 → ~921 Geçici Dosya
2. Adım: k = 1, çünkü work_mem=64 kB bir dosya kadar yer tutabiliyor

İterasyon Sayısı log₁ 921 = 921 İterasyon

Sonuç: Mevcut work_mem ile, harici sıralama **921 iterasyon** gerektirir.

❖ Sistemde yapılan harici sıralamada k-way merge de k değeri nedir?

work_mem = 64 kB.

Geçici dosya boyutları toplamı: 58,928 kB.

Harici sıralama yöntemi: external merge sort

K = work_mem / Bir Geçici Dosya Boyutu

Geçici Dosya Boyutu

Sorgunun plan çıktısına göre:

- Her işçi için geçici dosyalar yaklaşık 20 MB boyutunda:
 - Ana işlem: 20,416 kB.
 - İşçi 0: 19,328 kB.
 - İşçi 1: 19,184 kB.

Bu dosyalar sıralama işlemi sırasında eşit parçalara bölünür. Tek bir geçici dosyanın boyutunu yaklaşık 20,000 kB (20 MB) olarak kabul edebiliriz.

K = work_mem / Bir Geçici Dosya Boyutu = 64kB / 20 000kB = 0,0032

K değeri, mevcut work_mem değerine göre **1**'den küçük olduğu için, aynı anda sadece **1 geçici dosya** işlenebilir.

Sonuç: Bu sistemde yapılan harici sıralamada **k değeri = 1**'dir. Bu, sıralama işleminin her birleştirme aşamasında sadece bir dosya işlenebileceğini ve dolayısıyla **çok sayıda iterasyon** gerekeceğini gösterir. work_mem artırılarak daha yüksek bir k değeri elde edilebilir, böylece sıralama daha verimli hale gelir.