BSM211 Veritabanı Yönetim Sistemleri - Celal ÇEKEN, İsmail ÖZTEL, Veysel Harun ŞAHİN

Başarım Eniyileme (Performance Optimization)

- Örnekler İçin Pagila Veritabanı Kullanılmaktadır.

EXPLAIN ANALYSE

- EXPLAIN ANALYSE ifadesi ile SQL sorgularının başarımına ilişkin detaylı bilgi edinilir. Sorgunun nasıl çalıştırılacağını açıklayan sorgu planı döndürülür.
    - EXPLAIN ile pg_statistic katalog tablosuna dayalı olarak, başarımla ilgili tahmini değerler döndürülür (cost=0.00..16.49 rows=1 width=70).
        - İlgili tablo veya tabloların içeriğine dair istatistikler (kolon içerisinde en çok bulunan değer ve histogramı, distinct değerler, byte olarak ortalama genişlik, ortalama değer, null olanların sayısı, satır sayısı, vb.) "pg_statistic" içerisinde saklanır. Bu değerler "ANALYZE" işlemiyle güncellenebilir ve büyük tablolarda örnekleme yapılarak oluşturulabilir.
        - (cost=0.00..16.49 rows=1 width=70) örneğindeki değerlerin anlamları:
            - 0.00 : ilk satırın getirilme maliyeti (birimi "page cost": tek bir sayfanın getirilme maliyeti(zaman ve kaynak kullanımıyla ilgili))
            - 16.49 : tüm satırların getirilme maliyeti (birimi "page cost")
            - rows: getirilecek satır sayısı
            - width: bir satırın byte cinsinde ortalama genişliği
    - ANLYZE ile birlikte sorgu gerçekten çalıştırılarak gerçek değerler döndürülür (actual time=0.115..0.181 rows=1 loops=1). (INSERT, UPDATE, DELETE vb. işlemler için dikkatli kullanılmalıdır.)
        - (actual time=0.115..0.181 rows=1 loops=1) örneğindeki değerlerin anlamları:
            - 0.115 ms : ilk satırın getirilmesi için geçen zaman
            - 0.181 ms : tüm satırların getirilmesi için geçen zaman
            - rows: getirilen satır sayısı
            - loops: döngü sayısı Toplam zaman= 0.181x1

```
EXPLAIN ANALYSE
SELECT * FROM "customer";
------------------------------
Seq Scan on customer  (cost=0.00..14.99 rows=599 width=70) (actual
time=0.009..0.087 rows=599 loops=1)
Planning Time: 0.092 ms   // uygun planı seçme zamanı
Execution Time: 0.139 ms // Seçilen planın çalıştırılma zamanı. Actual Time
içerisine oturum açma/kapatma zamanı eklenerek bulunur
```

```
EXPLAIN ANALYSE
SELECT * FROM "customer"
```

```
WHERE "first_name" = 'Bruce';
------------------------------
Seq Scan on customer  (cost=0.00..16.49 rows=1 width=70) (actual time=0.115..0.181
rows=1 loops=1)
  Filter: ((first_name)::text = 'Bruce'::text)
  Rows Removed by Filter: 598
Planning Time: 0.110 ms
Execution Time: 0.202 ms
```

```
EXPLAIN ANALYSE
SELECT * FROM "customer"
WHERE "last_name" = 'Lee';
------------------------------
Index Scan using idx_last_name on customer  (cost=0.28..8.29 rows=1 width=70)
(actual time=0.035..0.037 rows=1 loops=1)
  Index Cond: ((last_name)::text = 'Lee'::text)
Planning Time: 0.127 ms
Execution Time: 0.062 ms
```

```
EXPLAIN ANALYSE
SELECT   "public"."customer"."customer_id",
         "public"."customer"."first_name",
         "public"."customer"."last_name",
         "public"."address"."phone"
FROM     "customer"
INNER JOIN "address"  ON "customer"."address_id" = "address"."address_id"
------------------------------
Hash Join  (cost=21.57..38.14 rows=599 width=29) (actual time=0.510..0.972
rows=599 loops=1)
  Hash Cond: (customer.address_id = address.address_id)
  ->  Seq Scan on customer  (cost=0.00..14.99 rows=599 width=19) (actual
time=0.009..0.097 rows=599 loops=1)
  ->  Hash  (cost=14.03..14.03 rows=603 width=16) (actual time=0.485..0.485
rows=603 loops=1)
        Buckets: 1024  Batches: 1  Memory Usage: 39kB
        ->  Seq Scan on address  (cost=0.00..14.03 rows=603 width=16) (actual
time=0.011..0.237 rows=603 loops=1)
Planning Time: 0.642 ms
Execution Time: 1.052 ms
```

PROJEKSİYON

- SELECT ifadesinde bütün alanlara projeksiyon yapmak (* kullanımı) yerine yalnızca gerekli olan alanlara projeksiyon yapmalıyız. Yani yalnızca gerekli alanların getirilmesini istemeliyiz. Böylece, işlem gecikmesi, iletim gecikmesi ve kaynak kullanımı azaltılmış olur.

```
EXPLAIN ANALYSE
SELECT *
FROM "customer"
INNER JOIN "store" ON "customer"."store_id" = "store"."store_id"
INNER JOIN "rental" ON "rental"."customer_id" = "customer"."customer_id"
INNER JOIN "inventory" ON "inventory"."store_id" = "store"."store_id"
INNER JOIN "film" ON "inventory"."film_id" = "film"."film_id";
```

- Execution time: 10968.823 ms

```
EXPLAIN ANALYSE
SELECT "customer"."first_name", "customer"."last_name",
    "film"."film_id", "film"."title"
FROM "customer"
INNER JOIN "store" ON "customer"."store_id" = "store"."store_id"
INNER JOIN "rental" ON "rental"."customer_id" = "customer"."customer_id"
INNER JOIN "inventory" ON "inventory"."store_id" = "store"."store_id"
INNER JOIN "film" ON "inventory"."film_id" = "film"."film_id";
```

- Execution time: 6220.990 ms

LIMIT ve OFFSET

```
EXPLAIN ANALYSE
SELECT "store"."store_id", "film"."title"
FROM "inventory"
INNER JOIN "film" ON "inventory"."film_id" = "film"."film_id"
INNER JOIN "store" ON "inventory"."store_id" = "store"."store_id";
```

- Execution time: 4.450 ms
- İlk 40 dan sonraki 20 kayıt getirilsin.

```
EXPLAIN ANALYSE
SELECT "store"."store_id", "film"."title"
FROM "inventory"
INNER JOIN "film" ON "inventory"."film_id" = "film"."film_id"
INNER JOIN "store" ON "inventory"."store_id" = "store"."store_id"
LIMIT 20 OFFSET 40;
```

- Execution time: 0.315 ms

```
EXPLAIN ANALYSE
SELECT "customer_id", "first_name", "last_name"
```

```
FROM "customer" ORDER BY "customer_id" DESC;
```

- Son 20 den sonraki 10 getirilsin

```
EXPLAIN ANALYSE
SELECT "customer_id", "first_name", "last_name"
FROM "customer" ORDER BY "customer_id" DESC
LIMIT 10 OFFSET 20;
```

SIRALAMA

- Gereksiz sıralama başarımı düşürür.

```
EXPLAIN ANALYSE
SELECT "store"."store_id", "film"."title"
FROM "inventory"
INNER JOIN "film" ON "inventory"."film_id" = "film"."film_id"
INNER JOIN "store" ON "inventory"."store_id" = "store"."store_id";
```

- Execution time: 4.968 ms

```
EXPLAIN ANALYSE
SELECT "store"."store_id", "film"."title"
FROM "inventory"
INNER JOIN "film" ON "inventory"."film_id" = "film"."film_id"
INNER JOIN "store" ON "inventory"."store_id" = "store"."store_id"
ORDER BY "film"."title";
```

- Execution time: 7.411 ms

INDEX

- Index olarak belirlenmiş alanlar üzerinde arama işlemi daha hızlı gerçekleştirilir.
- Aşağıdaki sorgularda “customer” tablosunun “last_name” alanı için index tanımlanmıştır.

```
EXPLAIN ANALYSE
SELECT * FROM "customer"
WHERE "first_name" = 'Jeniffer';
```

- Execution time: 0.132 ms

```

```

```
EXPLAIN ANALYSE
SELECT * FROM "customer"
WHERE "last_name" = 'Davis';
```

- Execution time: 0.036 ms
- Örnek Ek Veritabanı

```
CREATE DATABASE "TestVeritabani"
ENCODING='UTF-8'
LC_COLLATE='tr_TR.UTF-8'
LC_CTYPE='tr_TR.UTF-8'
OWNER postgres
TEMPLATE=template0;
```

- Windows işletim sistemi için

```
CREATE DATABASE "TestVeritabani"
ENCODING='UTF-8'
LC_COLLATE='Turkish_Turkey.1254'
LC_CTYPE='Turkish_Turkey.1254'
OWNER postgres
TEMPLATE=template0;
```

```
CREATE TABLE "Kisiler" (
    "kisiNo" SERIAL,
    "adi" VARCHAR(40) NOT NULL,
    "soyadi" VARCHAR(40) NOT NULL,
    "kayitTarihi" TIMESTAMP DEFAULT '2019-01-01 01:00:00',
    CONSTRAINT "urunlerPK1" PRIMARY KEY("kisiNo")
);
```

```
CREATE OR REPLACE FUNCTION "veriGir"(kayitSayisi integer)
RETURNS VOID
AS
$$
BEGIN
    IF kayitSayisi > 0 THEN
        FOR i IN 1 .. kayitSayisi LOOP
            insert into "Kisiler" ("adi","soyadi", "kayitTarihi")
            Values(
                substring('ABCÇDEFGĞHIiJKLMNOÖPRSŞTUÜVYZ' from
ceil(random()*10)::smallint for ceil(random()*20)::SMALLINT),
                substring('ABCÇDEFGĞHIiJKLMNOÖPRSŞTUÜVYZ' from
```

```
ceil(random()*10)::smallint for ceil(random()*20)::SMALLINT),
                NOW() + (random() * (NOW()+'365 days' - NOW()))
                 );
        END LOOP;
    END IF;
END;
$$
LANGUAGE 'plpgsql'  SECURITY DEFINER;
```

```
SELECT "veriGir"(100000);
```

```
EXPLAIN ANALYZE
SELECT * FROM "Kisiler"
WHERE "adi"='DENEME' -- Satırlardan birinin adi alanı "DENEME" olarak
değiştirilmeli
----------------------------------------
Seq Scan on "Kisiler"  (cost=0.00..2107.00 rows=496 width=38) (actual
time=20.214..20.215 rows=1 loops=1)
  Filter: ((adi)::text = 'DENEME'::text)
  Rows Removed by Filter: 99999
Planning Time: 0.085 ms
Execution Time: 20.237 ms
```

- Execution time: 20.237 ms

```
CREATE INDEX "adiINDEX" ON "public"."Kisiler" USING btree( "adi" Asc NULLS Last );
```

```
EXPLAIN ANALYZE
SELECT * FROM "Kisiler"
WHERE "adi"='DENEME' -- Satırlardan birinin adi alanı "DENEME" olarak
değiştirilmeli
----------------------------------------
Bitmap Heap Scan on "Kisiler"  (cost=12.26..784.32 rows=496 width=38) (actual
time=0.052..0.053 rows=1 loops=1)
  Recheck Cond: ((adi)::text = 'DENEME'::text)
  Heap Blocks: exact=1
  ->  Bitmap Index Scan on "adiINDEX"  (cost=0.00..12.14 rows=496 width=0) (actual
time=0.045..0.046 rows=1 loops=1)
        Index Cond: ((adi)::text = 'DENEME'::text)
Planning Time: 0.123 ms
Execution Time: 0.083 ms
```

- Execution time: 0.083 ms

```
EXPLAIN ANALYZE
SELECT * FROM "Kisiler"
WHERE "adi"!='DENEME' -- Satırlardan birinin adi alanı "DENEME" olarak
değiştirilmeli
-- Sorgu sonucu getirilecek satır çok fazla ise, sonuçlar indeks tablosu
kullanılmadan doğrudan veri tablosundan getirilir.
----------------------------------------
Seq Scan on "Kisiler"  (cost=0.00..2107.00 rows=99504 width=38) (actual
time=0.019..26.304 rows=99999 loops=1)
  Filter: ((adi)::text <> 'DENEME'::text)
  Rows Removed by Filter: 1
Planning Time: 0.088 ms
Execution Time: 32.305 ms
```

- Execution time: 32.305 ms

Birleşim (INNER JOIN), IN ve EXIST (İlintili Sorgu)

- İlintili sorgu, özellikle EXIST ifadesi ile birlikte, daha iyi sonuç verebilir.

```
EXPLAIN ANALYZE
SELECT DISTINCT "customer"."first_name", "customer"."last_name"
FROM "customer"
INNER JOIN "payment"
ON "payment"."customer_id" = "customer"."customer_id";
```

- Execution time: 18.817 ms

```
EXPLAIN ANALYZE
SELECT "customer"."first_name", "customer"."last_name"
FROM "customer"
WHERE "customer_id" IN (SELECT  "customer_id" FROM "payment");
```

- Execution time: 3.980 ms

```
EXPLAIN ANALYSE
SELECT "customer"."first_name", "customer"."last_name"
FROM "customer"
WHERE EXISTS
    (SELECT "customer_id" FROM "payment"
     WHERE "customer"."customer_id" = "payment"."customer_id");
```

- Execution time: 3.683 ms

HAVING

- HAVING ifadesi seçim işlemi yapılıp gruplandırma işlemi tamamlandıktan sonra filtreleme yapmak için kullanılır. Filtreyi, mümkünse gruplama işleminden önce eklemek başarımı artırır.

```
EXPLAIN ANALYSE
SELECT "category"."name", COUNT("film"."film_id")
FROM "film"
LEFT OUTER JOIN "film_category" ON "film"."film_id" = "film_category"."film_id"
LEFT OUTER JOIN "category" ON "film_category"."category_id" =
"category"."category_id"
GROUP BY "category"."name"
HAVING "category"."name" = 'Horror' OR "category"."name" = 'Comedy';
```

- Execution time: 0.922 ms

```
EXPLAIN ANALYSE
SELECT "category"."name", COUNT("film"."film_id")
FROM "film"
LEFT OUTER JOIN "film_category" ON "film"."film_id" = "film_category"."film_id"
LEFT OUTER JOIN "category" ON "film_category"."category_id" =
"category"."category_id"
WHERE "category"."name" = 'Horror' OR "category"."name" = 'Comedy'
GROUP BY "category"."name";
```

- Execution time: 0.898 ms

Alt Sorgu Sayısı

- Bazen ana sorguda birden fazla alt sorgu bulunabilir. Bu durumda alt sorgu bloklarının sayısını azaltmaya çalışmalıyız.
- Bu bölüm NorthWind veritabanı kullanmaktadır.

```
EXPLAIN ANALYSE
SELECT * FROM "products"
WHERE "UnitPrice" < (SELECT AVG("UnitPrice") FROM "products")
AND "UnitsInStock" < (SELECT AVG("UnitsInStock") FROM "products");
```

- 22:12:27 Query time: 2 millisecond(s), Number of cursor's records: 11

```
EXPLAIN ANALYSE
SELECT * FROM "products"
WHERE ("UnitPrice", "UnitsInStock") <
    (SELECT AVG("UnitPrice"), AVG("UnitsInStock") FROM "products");
```

- 22:12:32 Query time: 1 millisecond(s), Number of cursor's records: 8

UNION ve UNION ALL

- UNION yerine UNION ALL komutunu kullanmaya çalışmalıyız. UNION komutu icra edilirken DISTINCT işlemi de gerçekleştirildiği için daha yavaştır.

```
EXPLAIN ANALYSE
SELECT "rental_id" FROM "rental"
UNION
SELECT "rental_id" FROM "payment";
```

- Execution time: 20.382 ms

```
EXPLAIN ANALYSE
SELECT "rental_id" FROM "rental"
UNION ALL
SELECT "rental_id" FROM "payment";
```

- Execution time: 10.742 ms

WHERE

- WHERE koşul ifadeleri yazarken dikkat etmemiz gereken hususlar.

```
EXPLAIN ANALYSE
SELECT * FROM "film" WHERE SUBSTR("title", 2, 2) = 'la';
```

- 22:44:30 Query time: 2 millisecond(s), Number of affected records: 15

```
EXPLAIN ANALYSE
SELECT * FROM "film" WHERE "title" LIKE '_la%';
```

- 22:44:25 Query time: 1 millisecond(s), Number of affected records: 15

Genel Kurallar

- Belirli boyutu aşan ikili nesneleri (resim, pdf vb.) depolamak için, ilk olarak onları dosyalama sistemine yerleştiriniz ve sonrasında veritabanına dosyanın konumunu ekleyiniz.
- SQL standart kurallarını takip ediniz.

VACUUM ve ANALYSE

- PostgreSQL'de bir kayıt silindiği zaman aslında gerçekten silinmez.

- Yalnızca silindiğine ilişkin bir işaret olur. Dolayısıyla belirli bir süre sonra depolama alanı problemi oluşabilir. Silinen kayıtların gerçekten tablodan silinmesini gerçekleştirmek için VACUUM komutu kullanılır. Bu yapıldığında depolama alanımızda yer açılacaktır.
- ANALYSE işlemi sonucu, ilgili tablo veya tabloların içeriğine dair istatistikler (kolon içerisinde en çok bulunan değer ve histogramı, byte olarak ortalama genişlik, ortalama değer, null olanların sayısı vb.) "pg_statistic" sistem katalogunda güncellenmiş olarak saklanır. Daha sonra bu bilgi, sorgu planlayıcısının (query planner) sorguları en etkin şekilde nasıl çalıştıracağının belirlenmesi işleminde kullanılır.
- VACUUM ve ANALYSE işlemini, veritabanı kullanımının az olduğu zamanlarda, günde bir kez uygulamak sorgu hızını artırır.

VACUUM

- Seçili veritabanındaki tüm tablolara vacuum işlemi uygula.

```
VACUUM;
```

- Seçili veri tabanındaki tüm tablolara vacuum full işlemi uygula.
- Bu işlem daha uzun sürer. Tabloları kilitleyerek yeni bir kopyasını oluşturur ve daha sonra eski tabloyu siler.

```
VACUUM FULL;
```

- customer tablosuna vacuum işlemi uygula.

```
VACUUM "customer";
```

AUTOVACUUM Ayarı

- Eşik (threshold) değeri 5000 kayıt olsun. Bu eşik değerinin üzerine "eşik değeri * ölçek faktörü" de eklendikten sonra ulaşılan kayıt sayısı kadar güncelleme veya silme işlemi yapıldıktan sonra VACUUM işlemi başlatılır. Bu ayar postgresql.conf dosyasında da belirtilebilir.
- Varsayılan eşik değeri 50 kayıttır.

```
ALTER TABLE table_name
SET (autovacuum_vacuum_threshold = 5000);
```

- Eşik değerini %40 aştıktan sonra otomatik vakum işlemi yap. Bu ayar postgresql.conf dosyasında da belirtilebilir.
- Varsayılan ölçek faktörü (scale factor) 0.2'dir.

```
ALTER TABLE table_name
SET (autovacuum_vacuum_scale_factor = 0.4);
```

ANALYSE

- Seçili veritabanındaki tüm tablolara ANALYSE işlemi uygula.

```
ANALYSE;
```

- "payment" tablosuna ANALYSE işlemi uygula.

```
ANALYSE "payment";
```

```
SELECT * FROM "pg_statistic";
```

```
SELECT "relname", "last_vacuum", "last_autovacuum", "last_analyze",
"last_autoanalyze"
FROM "pg_stat_all_tables"
WHERE "schemaname" = 'public';
```