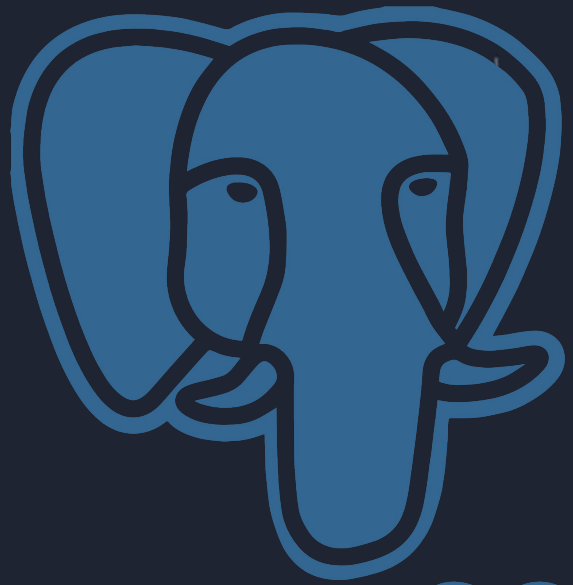

PostgreSQL 16 ile Gelen 17 ile Gelecek Olan Yenilikler

COOKSOFT
WWW.COOKSOFT.COM.TR

Şahap Aşçı - Cooksoft Teknoloji A.Ş.

PostgreSQL 16 ile Gelen Yenilikler

PostgreSQL 16, birden fazla önemli değişiklik ile geldi.

- Performans İyileştirmeleri
- Logical Replication
- Geliştirici Deneyimi
- Monitoring
- Erişim Kontrolü ve Güvenlik

Performans İyileştirmeleri

- **Sorgu planlayıcı optimizasyonu**

- FULL ve RIGHT (JOIN) birleştirmeleri paralelleştirebiliyor.

- DISTINCT veya ORDER BY cümlesi ile toplama işlevleri kullanan sorgular için daha iyi optimize edilmiş planlar oluşturabiliyor, SELECT DISTINCT sorguları için artımlı sıralamaları kullanabiliyor.

- Window functions daha verimli çalışacak şekilde optimize edebiliyor.

- Kullanıcıların birleştirilmiş bir tabloda bulunmayan satırları tanımlamasına olanak tanıyan RIGHT and OUTER "anti-joins" performansı da geliştirilmiştir.

ÖRNEK 1 : EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF) SELECT DISTINCT a,b FROM pg15_test order by a desc;

```
/ POSTGRESQL 15
                          QUERY PLAN
-----------------------------------------------------------------
 Unique (actual rows=1000000 loops=1)
   ->  Sort (actual rows=1000000 loops=1)
         Sort Key: a DESC, b
         Sort Method: external merge  Disk: 17664kB
         ->  Seq Scan on pg15_test (actual rows=1000000 loops=1)
 Planning Time: 0.106 ms
 Execution Time: 775.896 ms

/ POSTGRESQL 16
                           QUERY PLAN
-----------------------------------------------------------------
 Unique (actual rows=1000000 loops=1)
   ->  Incremental Sort (actual rows=1000000 loops=1)
         Sort Key: a DESC, b
         Presorted Key: a
         Full-sort Groups: 31250  Sort Method: quicksort  Average Memory: 26kB  Peak Memory: 26kB
         ->  Index Scan Backward using pg16_test_a_idx on pg16_test (actual rows=1000000 loops=1)
 Planning Time: 0.121 ms
 Execution Time: 537.181 ms
```

Karşılaştırmalı sorgu planı gösterimindeki işletim sistemleri ,kaynaklar ve data aynıdır.

ÖRNEK 2 :EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF, BUFFERS) SELECT a,COUNT(DISTINCT b) FROM pg15_aggtest GROUP BY a;

/ POSTGRESQL 15

```
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 GroupAggregate (actual rows=10 loops=1)
   Group Key: a
   Buffers: shared hit=892, temp read=4540 written=4560
   ->  Index Only Scan using pg15_aggtest_a_b_idx on pg15_aggtest (actual rows=1000000 loops=1)
         Heap Fetches: 0
         Buffers: shared hit=892
 Planning Time: 0.060 ms
 Execution Time: 764.638 ms
```

/ POSTGRESQL 16

```
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 GroupAggregate (actual rows=10 loops=1)
   Group Key: a
   Buffers: shared hit=892
   ->  Index Only Scan using pg16_aggtest_a_b_idx on pg16_aggtest (actual rows=1000000 loops=1)
         Heap Fetches: 0
         Buffers: shared hit=892
 Planning Time: 0.061 ms
 Execution Time: 260.074 ms
```

Karşılaştırmalı sorgu planı gösterimindeki işletim sistemleri ,kaynaklar ve data aynıdır.

ÖRNEK 3 : EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)

```
SELECT * FROM (SELECT * FROM t1 UNION ALL SELECT * FROM t2) t INNER JOIN lookup l ON l.a = t.a;
```

/ POSTGRESQL 15

```
                                QUERY PLAN
---------------------------------------------------------------------------
 Nested Loop (actual rows=2000000 loops=1)
   ->  Seq Scan on lookup l (actual rows=1000000 loops=1)
   ->  Append (actual rows=2 loops=1000000)
         ->  Index Only Scan using t1_pkey on t1 (actual rows=1 loops=1000000)
               Index Cond: (a = l.a)
               Heap Fetches: 1000000
         ->  Index Only Scan using t2_pkey on t2 (actual rows=1 loops=10000
               Index Cond: (a = l.a)
               Heap Fetches: 1000000
 Planning Time: 0.263 ms
 Execution Time: 4350.777 ms
```

/ POSTGRESQL 16

```
                                QUERY PLAN
---------------------------------------------------------------------------
 Nested Loop (actual rows=2000000 loops=1)
   ->  Seq Scan on lookup l (actual rows=1000000 loops=1)
   ->  Memoize (actual rows=2 loops=1000000)
         Cache Key: l.a
         Cache Mode: logical
         Hits: 999990  Misses: 10  Evictions: 0  Overflows: 0  Memory Usage: 2kB
         ->  Append (actual rows=2 loops=10)
               ->  Index Only Scan using t1_pkey on t1 (actual rows=1 loops=10)
                     Index Cond: (a = l.a)
                     Heap Fetches: 10
               ->  Index Only Scan using t2_pkey on t2 (actual rows=1 loops=10)
                     Index Cond: (a = l.a)
                     Heap Fetches: 10
 Planning Time: 0.329 ms
 Execution Time: 906.610 ms
```

ÖRNEK 4 : `EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF) SELECT * FROM (SELECT id,ROW_NUMBER() OVER (ORDER BY score) rn,score FROM scores) m WHERE rn <= 10;`

/ POSTGRESQL 15

```
                                QUERY PLAN
---------------------------------------------------------------------------
 WindowAgg (actual rows=10 loops=1)
   Run Condition: (row_number() OVER (?) <= 10)
   ->  Index Scan using scores_score_idx on scores (actual rows=49989 loops=1)
 Planning Time: 0.375 ms
 Execution Time: 29.676 ms
(5 rows)
```

/ POSTGRESQL 16

```
                                QUERY PLAN
---------------------------------------------------------------------------
 WindowAgg (actual rows=10 loops=1)
   Run Condition: (row_number() OVER (?) <= 10)
   ->  Index Scan using scores_score_idx on scores (actual rows=11 loops=1)
 Planning Time: 0.199 ms
 Execution Time: 0.106 ms
(5 rows)
```

örnekteki geliştirme
- row_number()
- rank()
- dense_rank()
- percent_rank()
- cume_dist()
- ntile() cümlecikleri için benzerdir.

Karşılaştırmalı sorgu planı gösterimindeki işletim sistemleri ,kaynaklar ve data aynıdır.

ÖRNEK 4 : `EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF) SELECT nt.* FROM normal_table nt LEFT JOIN part_tab pt ON nt.part_tab_id = pt.id;`

/ POSTGRESQL 15

```
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 Merge Right Join (actual rows=0 loops=1)
   Merge Cond: (pt.id = nt.part_tab_id)
   ->  Merge Append (actual rows=0 loops=1)
         Sort Key: pt.id
         ->  Index Only Scan using part_tab_p0_pkey on part_tab_p0 pt_1 (actual rows=0 loops=1)
               Heap Fetches: 0
         ->  Index Only Scan using part_tab_p1_pkey on part_tab_p1 pt_2 (actual rows=0 loops=1)
               Heap Fetches: 0
   ->  Sort (actual rows=0 loops=1)
         Sort Key: nt.part_tab_id
         Sort Method: quicksort  Memory: 25kB
         ->  Seq Scan on normal_table nt (actual rows=0 loops=1)
 Planning Time: 1.144 ms
 Execution Time: 0.060 ms
```

Bu örnekte normal bir tabloya partition table' ın join yapıldığı görülmüştür.

/ POSTGRESQL 16

```
                    QUERY PLAN
---------------------------------------------------
 Seq Scan on normal_table nt (actual rows=0 loops=1)
 Planning Time: 0.167 ms
 Execution Time: 0.012 ms
```

- **Explain (GENERIC PLAN) Parametresi**

PostgreSQL 16 ile EXPLAIN planındaki GENERIC_PLAN seçeneği eklenmiştir, Genel sorgu planını görüntülemek için spesifik değerler vermeden genel parametreler ile sorgu yürütme planı oluşturmaya olanak tanır.

```
postgres=# EXPLAIN (GENERIC_PLAN ON) SELECT * FROM pg_class WHERE relname=$1;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 Index Scan using pg_class_relname_nsp_index on pg_class  (cost=0.27..2.49 rows=1 width=273)
   Index Cond: (relname = $1)
(2 rows)

postgres=# EXPLAIN (GENERIC_PLAN ON) select * from pg_stat_io where stats_reset between $1 and $2;
                             QUERY PLAN
---------------------------------------------------------------------
 Function Scan on pg_stat_get_io b  (cost=0.00..0.45 rows=1 width=216)
   Filter: ((stats_reset >= $1) AND (stats_reset <= $2))
(2 rows)
```

- **Bulk loading**

  - COPY kullanarak hem tekli hem de eş zamanlı işlemler için iyileştirmeler içeriyor. (wait_event_type = ‘Lock’ AND wait_event = ‘extend’)
  - CREATE SUBSCRIPTION … WITH (binary = TRUE)

- **Vacuum performans**

  - Vacuum komutu freeze ile çalıştırılmasa bile uygun olduğu durumlarda freezing yapılması.

- **libpq yük dengelemesi**

- Postgresql 16 ile beraber, psql gibi libpq kullanan araçlarda artık yük dengeleme desteği eklenmiştir. load_balance_hosts parametresi eklenmiştir.

```
+ Cluster: pg16-test (7327314495227931738) -----------+----+-----------+
| Member        | Host             | Role    | State     | TL | Lag in MB |
+---------------+------------------+---------+-----------+----+-----------+
|   patroni01   | 192.168.130.170 | Leader  | running   |  4 |           |
|   patroni02   | 192.168.130.171 | Replica | streaming |  4 |         0 |
|   patroni03   | 192.168.130.172 | Replica | streaming |  4 |         0 |
+---------------+------------------+---------+-----------+----+-----------+
```

- Test için örnek bir patroni ortamı oluşturulmuştur.

```
postgres
      export PGDATABASE=cooksoft
      export PGHOSTS=192.168.130.170,192.168.130.171,192.168.130.172
      export PGUSER=cooksoft_owner
      export PGPASSWORD=pg16test

 └─$  psql -h $PGHOSTS -c "select inet_server_addr()" 'load_balance_hosts=random'
 inet_server_addr
------------------
 192.168.130.172
```

- Connection’ın random iplere yönlendirildiği görülmektedir.

```
 └─$  psql -h $PGHOSTS -c "select inet_server_addr()" 'load_balance_hosts=random
target_session_attrs=standby'
 inet_server_addr
------------------
 192.168.130.171
```

- Connection’ın cluster standby iplerine yönlendirildiği görülmektedir.

target_session_attrs:

- any (default)
- read-write
- read-only
- primary
- standby
- prefer-standby

- **JAVA**

```
jdbc:postgresql://host1:port1,host2:port2/database

jdbc:postgresql://node1,node2,node3/accounting?targetServerType=primary

jdbc:postgresql://node1,node2,node3/accounting?targetServerType=preferSecondary&loadBalanceHosts=true
```

- **.Net**

```
Host=server1,server2,server3;Username=test;Password=test;Load Balance Hosts=true;Target Session
Attributes=prefer-standby

dataSource.WithTargetSession(TargetSessionAttributes.Primary)

dataSource.WithTargetSession(TargetSessionAttributes.PreferStandby);
```

PostgreSQL 16 ile Standby Üzerinden Logical Replikasyon

PostgreSQL 16, standby sunuculardan logical replikasyon gerçekleştirme yeteneğini tanıtarak bir dizi fayda sağlar:

- Salt okunur bir sunucuda mantıksal kodlamayı yaratma
- Birincil sunucudaki iş yükünü azaltma
- Çoklu sistemler arasında veri senkronizasyonunu gerektiren uygulamalar ve denetim amaçları için yüksek erişilebilirlik sağlama
- Salt okunur replikasyon slotları, birincil sunucu arızalarında standby abonelerine kesintisiz hizmet sağlayarak sürekliliği sağlar.
- Büyük veri işlemleri, veri depolama, analitik, veri entegrasyonu ve iş zekası için esneklik sağlar
- Origin parametresi ile bi-directional replikasyon imkanı

PostgreSQL 16 “reserved_connections” Parametresi

- Bağlantı yuvalarının sayısını belirler.
- pg_use_reserved_connections rol ayrıcalığına sahip roller için ayrılmış olan bağlantı yuvalarının.

Neden Önemli?

- PostgreSQL veritabanına yapılan bağlantılar için kısıtlama sağlar.
- Özellikle yüksek trafikli sistemlerde veritabanı kaynaklarını etkili bir şekilde yönetir.

İşlevi:

- Bağlantı yuvalarını belirli roller için ayrılmıştır.
- Yeni bağlantılar bu yuvaların dışında ise, sadece süper kullanıcılar veya **pg_use_reserved_connections** ayrıcalığına sahip roller bağlanabilir.

```
postgres=# \dconfig *(reserved|max)_connections
      List of configuration parameters
            Parameter           | Value
--------------------------------+-------
 max_connections                | 100
 reserved_connections           | 0
 superuser_reserved_connections | 3
```

Yanda max_connection sayısı, ayrılan reserved_connection sayısı ve superuser_reserved_connection sayısı görülmektedir.

```
postgres=# grant pg_use_reserved_connections to test_user;
GRANT ROLE
postgres=# \drg
                        List of role grants
 Role name |          Member of          |   Options    | Grantor
-----------+-----------------------------+--------------+----------
 test_user | pg_use_reserved_connections | INHERIT, SET | postgres
(1 row)
```

Yukarıda test_user isimli kullanıcımıza yetki tanımlıyoruz. Bu sayede normal kullanıcılar connection limitini doldurduklarında rezerve edilen connection'ı kullanarak bağlantı sağlayabilecek.

```
root@vivian:~# psql --host=192.168.120.217 -p 5433 -d test -U deneme;
Password for user deneme:
psql: error: connection to server at "192.168.120.217", port 5433 failed: FATAL:  remaining connection
slots are reserved for roles with privileges of the "pg_use_reserved_connections" role
root@vivian:~# psql --host=192.168.120.217 -p 5433 -d test -U test_user;
Password for user test_user:
psql (16.2 (Ubuntu 16.2-1.pgdg23.10+1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off)
Type "help" for help.
test=>
```

Yukarıda reserve connection yetkisi olmayan kullanıcı ile bağlanmayı denediğimizde connection dolu olduğu ve yetkimiz olmadığı için hata alıyoruz.

Aynı şekilde yetkisi olan kullanıcı ile bağlanmayı denediğimizde rezerve edilen connection ile bağlantı kuarabildik.

Performans ve İzleme İyileştirmeleri

Veritabanı performansını ayarlamak, herhangi bir veritabanı yöneticisi için kritik bir görevdir. Bu bağlamda, I/O işlemlerinin sistem üzerindeki etkisini anlamak büyük önem taşır. PostgreSQL 16, bu ihtiyaca cevap vermek için çeşitli önemli gelişmeler sunmaktadır.

- I/O Performansı Analizi için pg_stat_io
- Tablo ve Index İstatistiklerine erişim zamanı eklenmesi
- auto_explain İyileştirmesi
- Sorgu İzleme Doğruluğunun Arttırılması

- **I/O Performansı Analizi için pg_stat_io**
  PostgreSQL 16, I/O erişim modellerinin ayrıntılı analizi için yeni bir temel I/O ölçüm kaynağı olan pg_stat_io'yu getirmiştir. Bu özellik, veritabanı iş yüklerinin performansını optimize etmede kritik bir rol oynar.

- **Zaman Damgası Ekleme**
  pg_stat_all_tables (last_seq_scan , last_idx_scan)
  pg_stat_all_indexes (last_idx_scan)

```
postgres=# SELECT * FROM pg_stat_io;
```

backend_type	object	context	reads	read_time	writes	write_time	writebacks	writeback_time	extends	extend_time	op_bytes	hits	evictions	reuses	fsyncs	fsync_time	stats_reset
autovacuum launcher	relation	bulkread	0	0	0	0	0	0			8192	0	0	0			2024-04-04 09:25:51.762561+03
autovacuum launcher	relation	normal	3	0	0	0	0	0			8192	792	0		0	0	2024-04-04 09:25:51.762561+03
autovacuum worker	relation	bulkread	0	0	0	0	0	0			8192	0	0	0			2024-04-04 09:25:51.762561+03
autovacuum worker	relation	normal	240	0	0	0	0	0	29	0	8192	164604	0		0	0	2024-04-04 09:25:51.762561+03
autovacuum worker	relation	vacuum	88	0	0	0	0	0	0	0	8192	5855	0	49			2024-04-04 09:25:51.762561+03
client backend	relation	bulkread	854	0	0	0	0	0			8192	85	0	112			2024-04-04 09:25:51.762561+03
client backend	relation	bulkwrite	0	0	0	0	0	0	1344	0	8192	1056	0	0			2024-04-04 09:25:51.762561+03
client backend	relation	normal	2243	0	0	0	0	0	2782	0	8192	1289019	0		0	0	2024-04-04 09:25:51.762561+03
client backend	relation	vacuum	0	0	0	0	0	0	0	0	8192	0	0	0			2024-04-04 09:25:51.762561+03
client backend	temp relation	normal	0	0	0	0			0	0	8192	0	0				2024-04-04 09:25:51.762561+03
background worker	relation	bulkread	0	0	0	0	0	0			8192	0	0	0			2024-04-04 09:25:51.762561+03

- **auto_explain İyileştirmesi**
  PostgreSQL 16, parametrelendirilmiş ifadelere aktarılan değerleri günlüğe kaydederek auto_explain'i daha okunabilir hale getirir. Bu güncelleme, sorgu performansını izlerken daha ayrıntılı bilgi sağlar ve sorun giderme süreçlerini kolaylaştırır.

- **Sorgu İzleme Doğruluğunun Arttırılması**
  PostgreSQL 16, pg_stat_statements ve pg_stat_activity tarafından kullanılan sorgu izleme algoritmasının doğruluğunu artırır. Bu iyileştirme, veritabanı performansını izlemeyi ve optimize etmeyi daha güvenilir hale getirir.

PostgreSQL 17 ile Gelecek Yenilikler

Planlanan Çıkış Tarihi: 14 Kasım 2024

PostgreSQL 17 ile Gelecek Yenilikler

- incremental backups
- pg_combinebackup
- WAL summarizer process
- Logical Replication iyileştirmesi ve pg_replication_slots'a yeni alanlar
- pg_maintain predefined role

PostgreSQL 17’de incremental backups

- Core PostgreSQL'de artımlı yedekleme seçeneği eklendi.
- pg_basebackup çalıştırırken artımlı yedekleme seçeneği mevcut.
- Artımlı yedekler, tam yedeklemeyi referans alan ve değişen veri dosyalarının sadece değişen kısımlarını içeren yedeklemelerdir.

Avantajları:

- WAL (Write Ahead Log) ile farklıdır.
- WAL ile geri alma işleminde tüm değişikliklerin uygulanması gerekirken, artımlı yedeklerde sadece değişen kısımlar saklanır.
- Veritabanında sık sık güncelleme yapılıyorsa daha verimli bir geri alma sağlar.

pg_combinebackup

- pg_combinebackup, incremental bir yedeklemeden ve bağlı olduğu önceki yedeklemelerden tam yedeklemeyi yeniden oluşturmak için kullanılır.

WAL summarizer process

- Artımlı yedeklemelerin çalışması için önemli bir bileşen.
- Değişen veri dizinlerini belirlemek için kullanılır.
- Artımlı yedekleme işlemi sırasında bu bilgiler kullanılır.
- Opsiyonel bir özellik olup, etkinleştirilmesi gerekir.

PostgreSQL’de incremental backups İhtiyaç Nedenleri

- Veritabanı boyutu artsa bile veri dosyalarının büyük kısmı genelde değişmez.
- Yedeklenen veri boyutunu azaltır, geri alma işlemlerini hızlandırır.
- Yüksek WAL üretimi durumunda önemli bir zaman kazancı sağlar.

pg_replication_slots view’ına yeni alanlar eklendi

- inactive_since
- invalidation_reason (conflicting = true)
- failover
  - pg_create_logical_replication_slot (slot_name, plugin_name, … failover)
- synced

pg_maintain role

pg_maintain rolü ön tanımlı olarak gelecek ve aşağıdaki yetkilere sahip olacak;

- VACUUM
- ANALYZE
- REINDEX
- REFRESH MATERIALIZE VIEW
- CLUSTER
- LOCK TABLE

komutlarını tüm tablolar için çalıştırabilir.

İlginiz İçin Teşekkürler