TEMEL KONTROLLERİN ÖTESİNDE:

POSTGRESQL HEALTH CHECK'E DERİN BAKIŞ

DataPrime
PRIMARY SERVICE

PostgreSQL Health Check Yaklaşımı

PostgreSQL veri tabanı sistemleri kontrolleri iki ana başlık altında incelenmektedir:

- İşletim sistemi seviyesi kontroller
- Veri tabanı seviyesi kontroller

İŞLETİM SİSTEMİ

SEVİYESİ KONTROLLERİ

Disk ve Filesystem Yapısı

- Data / Wal / Tablespace ayrımı (Disk I/O çakışmasını azaltır)
- LVM tabanlı disk yönetimi (Disk büyütme ve esnek kapasite yönetimi)
- Mount bazlı monitoring (Kapasite ve performans takibini kolaylaştırma)
- Ayrı archive alanı (WAL archive büyümelerinin data alanını etkilememesi)
- Tablespace ayrımı (Yoğun objelerin farklı storage sistemlerine taşınabilmesi)
- Ayrı log dizini (Troubleshooting süreçlerini kolaylaştırma)

Bellek Yönetimi

- **Swappiness**: Linux kernel'in RAM dolmaya başladığında nasıl davranacağını belirleyen parametredir. (varsayılan değer 60)
- Varsayılan değer RAM'de hâlâ boş alan varken Kernel'in PostgreSQL process belleğini (shared_buffers dahil) diske taşıyabileceği anlamına gelir.
- Tavsiye edilen değer: vm.swappiness = 1

Bellek Yönetimi

Neden 1 Öneriyoruz ? (vm.swappiness = 1)

•Swap son çare olarak kullanılabilir.

•OOM öncesi sistem davranışı gözlemlenebilir.

•Ani process kill riski azalır.

Neden 0 Önermiyoruz ? (vm.swappiness = 0)

•Kernel swap kullanmaktan tamamen kaçınır.

•Bellek gerçekten tükendiğinde OOM killer devreye girer ve doğrudan PostgreSQL'i öldürebilir.

Açık Dosya Limiti (Open File Descriptors)

PostgreSQL'de her veritabanı dosyası, WAL segmenti, soket ve pipe için bir dosya tanımlayıcı (fd) kullanılır:

- Tabloları, indeksleri ve sistem kataloglarını açma
- Soketler aracılığıyla istemci bağlantılarını yönetme
- WAL (Yazma Öncesi Günlük) dosyalarını yönetme
- Sıralama veya karma oluşturma için geçici dosyalarla çalışma

- Limit aşılınca 'Too many open files' hatasıyla bağlantılar reddedilir veya veri dosyaları açılamaz.

Mevcut durumu kontrol et:

1 Sistem geneli limit (kernel)

```
cat /proc/sys/fs/file-max
```

```
9223372036854775807
```

Sistem genelinde aynı anda açılabilecek toplam file descriptor kapasitesi.
Modern sistemlerde bu değer çok yüksek (2^63-1) olabilir ve genellikle sorun oluşturmaz.

2 PostgreSQL process'inin anlık limiti

```
cat /proc/$(pgrep -x postgres | head -1)/limits | grep -i "open files"
```

```
Max open files    65536    65536  files
```

Soft limit	:	Aktif kullanılan limit
Hard limit	:	Çıkılabilecek maksimum limit

3 PostgreSQL'in şu an kaç fd kullandığı

```
ls /proc/$(pgrep -x postgres | head -1)/fd \
2>/dev/null | wc -l
```

```
2487
```

PostgreSQL process'inin şu anda açık tuttuğu file descriptor (fd) sayısı.
Data file'lar, WAL, socket'ler, temp file'lar vb. dahildir.

4 systemd ile yönetiliyorsa (çoğu modern sistem)

```
systemctl show postgresql | grep -i limitnofile
```

```
LimitNOFILE=65536
```

systemd, PostgreSQL servisine bu kadar fd açma hakkı vermiş.
Gerçek limit genellikle buradan gelir.

5 ulimit (o oturumun limiti — servis limiti farklı olabilir)

```
ulimit -n
```

```
1024
```

Bu değer bulunduğun shell oturumunun limitidir. PostgreSQL servisinin gerçek limiti olmayabilir (özellikle systemd ile yönetiliyorsa).

Ayarlama — 3 katman:

Katman 1: Sistem geneli (kernel)

▶ **Mevcut değeri gör**

```
cat /proc/syss/fs/file-max
```

```
9223372036854775807
```

▶ **Artır (örn. 1 milyon)**

```
echo "fs.file-max = 1000000" >> /etc/sysctl.d/99-postgresql.conf
sysctl -p /etc/sysctl.d/99-postgresql.conf
```

Modern sistemlerde bu değer zaten çok yüksek olabilir.
Yine de kontrol edip, gerekiyorsa artırabilirsiniz.

Katman 2: Kullanıcı limitleri (/etc/security/limits.conf)

▶ **/etc/security/limits.conf veya /etc/security/limits.d/postgresql.conf dosyasına ekle**

```
postgres    soft    nofile    65536
postgres    hard    nofile    65536
```

PAM tabanlı oturumlarda geçerlidir.
systemd ile yönetilen servislerde bu ayar etkisizdir.
Katman 3 zorunludur.

Katman 3: systemd servis limiti (modern sistemlerde şart)

▶ **systemd override oluştur**

```
mkdir -p /etc/systemd/system/postgresql.service.d/
cat > /etc/systemd/system/postgresql.service.d/limits.conf << 'EOF'
[Service]
LimitNOFILE=65536
EOF
```

Uygula

1 sysctl ayarını yükle (Katman 1 için)

```
sysctl -p /etc/sysctl.d/99-postgresql.conf
```

2 systemd yeniden yükle

```
systemctl daemon-reload
```

3 PostgreSQL servisini yeniden başlat

```
systemctl restart postgresql
```

Doğrula

1 systemd limiti

```
systemctl show postgresql | grep LimitNOFILE
```

```
LimitNOFILE=65536
```

2 PostgreSQL process limiti

```
cat /proc/$(pgrep -x postgres | head -1)/limits |
| grep "open files"
```

```
Max open files    65536    65536  files
```

Beklenen: soft limit = hard limit = 65536 (veya daha büyük)
PostgreSQL dokümantasyonu minimum 1000 önerir; production için 65536 standarttır.

Önerilen değerler:

	Ortam	nofile değeri	Açıklama
	Küçük / test	16384	Az tablo, az bağlantı
	Orta ölçek	65536	Çoğu production için yeterli
	Büyük / çok tablolu	524288	Binlerce tablo veya partition

Hesaplama rehberi:
max_connections × (tablo başına dosya) + wal_segmentleri + sistem payı
≈ 500 × 3 + 100 + 1000 ≈ 3000 (minimum)
Güvenli üst sınır: 65536

Static Huge Pages (HugeTLB)

KONFİGÜRASYON ÖRNEĞİ

Linux (Önceden Ayırma)

```
# /etc/sysctl.conf
vm.nr_hugepages = 4096
# 4096 x 2MB = 8GB
```

PostgreSQL

```
huge_pages = try
# veya production'da
huge_pages = on
```

Huge Page küçük bellek alanlarının daha büyük parçalar halinde kullanılmasını sağlayan bir memory optimizasyonudur.

4kb'lık küçük bellek alanlarını kullanmak yerine büyük shared memory alanlarını daha az sayıda bellek sayfasıyla yönetmesini, CPU ve bellek yönetimi maliyetini azaltması için bu ayarı Linux işletim sisteminde düzenliyoruz ve veri tabanında konfigürasyonunu açıyoruz.

NASIL ÇALIŞIR?

1. Önceden Rezerve

OS, boot sırasında belirlenen sayıda 2MB'lık Huge Page'leri ayırır.

2. PostgreSQL Kullanır

PostgreSQL, shared memory alanlarını bu Huge Page'ler üzerinden kullanır.

3. Stabil ve Verimli Çalışma

Daha az page table, daha az TLB miss ve daha düşük CPU/memory overhead.

Static Huge Pages

- Manuel ve kontrollü tahsis
- Öngörülebilir performans
- Daha az CPU overhead
- Daha az TLB miss
- Büyük bellek alanlarında daha iyi ölçeklenebilirlik

NORMAL PAGES (4 KB)

8 GB shared_buffers
≈ 2,097,152 page (4 KB)

Büyük Page Table
Daha fazla memory overhead

Daha fazla TLB Miss
Daha yüksek CPU maliyeti

VS

HUGE PAGES (2 MB)

8 GB shared_buffers
≈ 4,096 page (2 MB)

Küçük Page Table
Daha az memory overhead

Daha az TLB Miss
Daha düşük CPU maliyeti

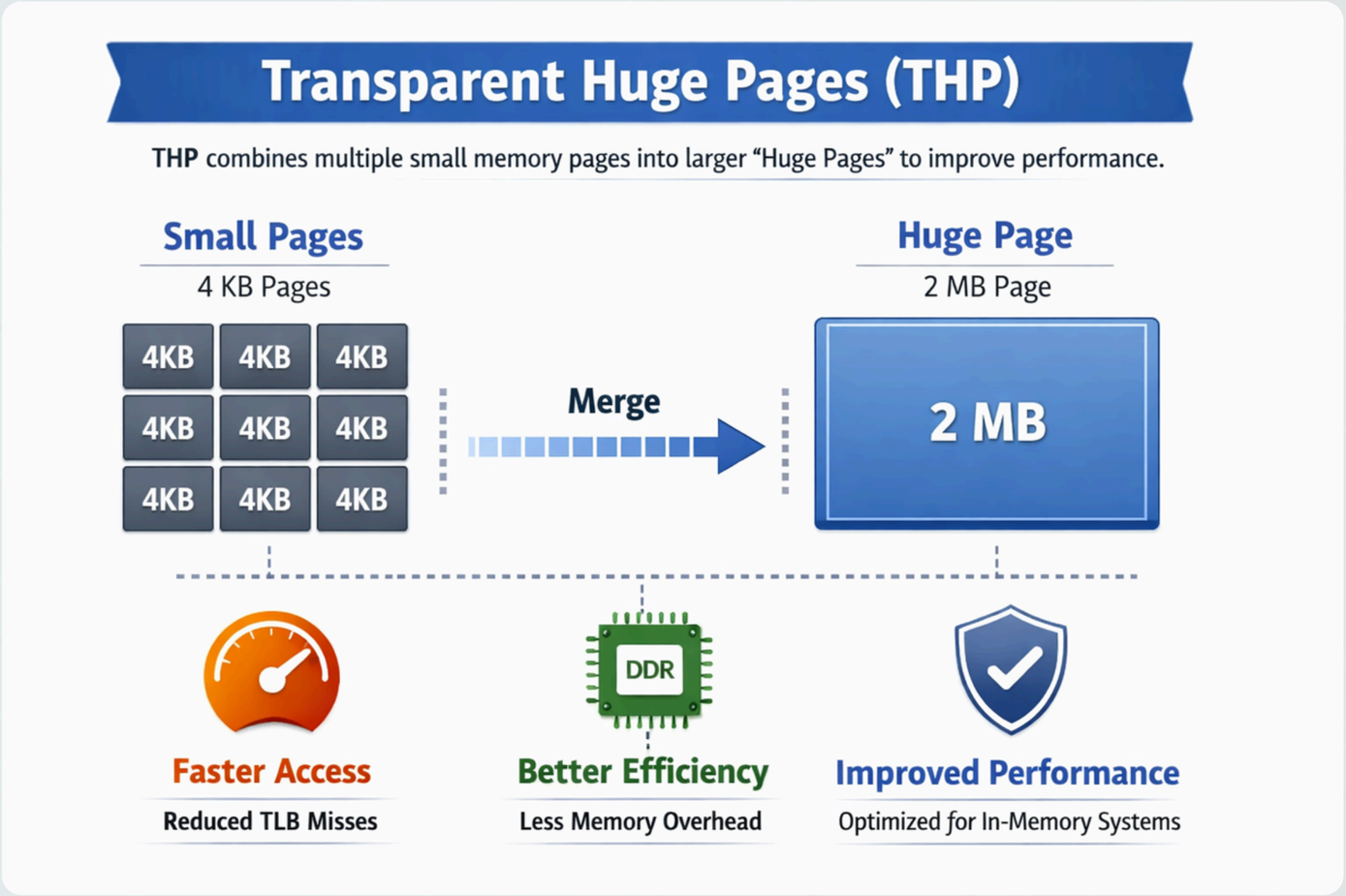
Transparent Huge Pages (THP)

THP, Linux kernel'in küçük bellek sayfalarını otomatik olarak 2 MB'lık büyük sayfalara dönüştürme mekanizmasıdır.

PostgreSQL için önerilmez çünkü:

- Otomatik page merge / defrag yapabilir
- Ani latency spike'ları oluşturabilir
- CPU kullanımında dalgalanma yaratabilir
- Performans öngörülebilirliğini bozabilir

Öneri:
Production PostgreSQL sistemlerinde THP kapatılmalıdır.



Dirty Page Yazma Ayarları

PostgreSQL'de bir page değiştiğinde, değişiklik önce shared_buffers içinde yapılır ve bu page dirty buffer olur.

Checkpoint bu dirty buffer'ı Linux'a gönderir. Ancak veri her zaman hemen fiziksel diske yazılmaz; Linux bunu bir süre page cache içinde dirty page olarak tutabilir.

.Linux çok fazla dirty page biriktirirse, bunları bir anda diske yazmaya çalışabilir. Bu da PostgreSQL'de I/O spike ve ani gecikmelere yol açabilir.

Amaç:
Büyük I/O spike yerine kontrollü writeback.

VERİ TABANI SEVİYESİ KONTROLLERİ

Temel Bağlantı Testleri

```
root@ubuntu1:~# ps -ef |grep postgres
postgres     735       1  0 10:24 ?        00:00:01 /usr/bin/python3 /usr/local/bin/patroni /etc/patroni/patroni.yml
postgres    1330       1  0 10:29 ?        00:00:00 postgres -D /data/patroni --config-file=/data/patroni/postgresql.conf --liste
n_addresses=192.168.56.183 --port=5432 --cluster_name=clusterpg --wal_level=replica --hot_standby=on --max_connections=100 --max_
wal_senders=10 --max_prepared_transactions=0 --max_locks_per_transaction=64 --track_commit_timestamp=off --max_replication_slots=
10 --max_worker_processes=8 --wal_log_hints=on
postgres    1332    1330  0 10:29 ?        00:00:00 postgres: clusterpg: logger
postgres    1333    1330  0 10:29 ?        00:00:00 postgres: clusterpg: checkpointer
postgres    1334    1330  0 10:29 ?        00:00:00 postgres: clusterpg: background writer
postgres    1341    1330  0 10:29 ?        00:00:00 postgres: clusterpg: postgres postgres 192.168.56.183(57122) idle
postgres    1348    1330  0 10:29 ?        00:00:00 postgres: clusterpg: walwriter
postgres    1349    1330  0 10:29 ?        00:00:00 postgres: clusterpg: autovacuum launcher
postgres    1350    1330  0 10:29 ?        00:00:00 postgres: clusterpg: logical replication launcher
postgres    1353    1330  0 10:29 ?        00:00:00 postgres: clusterpg: walsender replicator 192.168.56.184(56682) streaming 0/2
30001B8
root        1368    1285  0 10:31 pts/2    00:00:00 grep --color=auto postgres
```

```
postgres@ubuntu1:~$ pg_isready -h 192.168.56.183 -p 5432 -U postgres
192.168.56.183:5432 - accepting connections
postgres@ubuntu1:~$
postgres@ubuntu1:~$ psql -U postgres -c "SELECT version();"
                                                 version

------------------------------------------------------------------------------------------------------------------------
------
 PostgreSQL 16.13 (Ubuntu 16.13-1.pgdg24.04+1) on x86_64-pc-linux-gnu, compiled by gcc (Ubuntu 13.3.0-6ubuntu2~24.04.1) 13.3.0, 6
4-bit
(1 row)

postgres@ubuntu1:~$ psql -U postgres -c "SELECT now(), pg_postmaster_start_time(), now() - pg_postmaster_start_time() AS uptime;"
              now              |   pg_postmaster_start_time    |     uptime
-------------------------------+-------------------------------+-----------------
 2026-05-09 10:35:14.080733+00 | 2026-05-09 10:29:17.851639+00 | 00:05:56.229094
(1 row)
```

Veritabanı, Tablo, Tablespace Boyutlarının Kontrolü

- Veritabanı büyüme trendinin analiz edilmesi
- En büyük tablo ve index'lerin tespit edilmesi
- Tablespace kullanım dağılımının incelenmesi
- Kapasite planlama ve disk tüketim analizi
- Beklenmeyen veri büyümelerinin tespiti
- Kritik objelerin storage dağılımının değerlendirilmesi
- Backup süresi ve restore etkilerinin değerlendirilmesi

Veritabanı, Tablo ve Tablespace Boyutları

Kapasite kullanımı, büyüme trendleri ve alan dağılımını izlemek için temel sorgular.

Tablespace Boyutları

Veritabanındaki tablespace'lerin toplam kullanım alanını gösterir.

```
SELECT spcname as "tablespace name",
    pg_size_pretty(pg_tablespace_size(spcname)) AS size
FROM pg_tablespace;
```

Tablo ve Index Boyutları

Her tablonun toplam, tablo ve index boyutlarını listeler.

```
SELECT schemaname, tablename,
   pg_size_pretty(pg_total_relation_size(schemaname||'.'||tablename)) AS total_size,
   pg_size_pretty(pg_relation_size(schemaname||'.'||tablename)) AS table_size,
   pg_size_pretty(pg_indexes_size(schemaname||'.'||tablename)) AS index_size,
   round(pg_indexes_size(schemaname||'.'||tablename)::numeric /
      nullif(pg_total_relation_size(schemaname||'.'||tablename), 0) * 100, 1)
   AS index_pct
FROM pg_tables
WHERE schemaname NOT IN ('pg_catalog','information_schema')
ORDER BY pg_total_relation_size(schemaname||'.'||tablename) DESC
LIMIT 20;
```

Veritabanı Boyutları

Sistemdeki veritabanlarının toplam boyutlarını gösterir.

```
SELECT datname,
   pg_size_pretty(pg_database_size(datname)) AS size,
   pg_database_size(datname) AS size_bytes
FROM pg_database
ORDER BY pg_database_size(datname) DESC;
```

Tablespace Kullanımı

Tablespace Name	Size	Size (MB)	Size (GB)
pg_default	247 MB	247.00	0.24
pg_global	565 kB	0.55	0.00
TOPLAM	**247 MB**	**247.55**	**0.24**

En Büyük Tablolar

Schema	Table Name	Total Size	Table Size	Index Size	Index %
public	test02	42 MB	41 MB	0 bytes	0.0%
public	test03	17 MB	17 MB	0 bytes	0.0%
public	test01	4,992 kB	4,960 kB	0 bytes	0.0%

Index yüzdesi yüksek tablolar, yazma performansı ve disk kullanımı açısından izlenmelidir.

Veritabanı Kullanımı

Database Name	Size	Size (MB)	Size (GB)	Size (Bytes)
dvdrental	84 MB	84.00	0.08	88,046,615
testdb	77 MB	77.00	0.08	80,460,823
postgres	71 MB	71.00	0.07	74,677,271
template1	7,727 kB	7.54	0.01	7,912,471
template0	7,505 kB	7.33	0.01	7,684,623
TOPLAM	**317 MB**	**316.87**	**0.31**	**258,781,803**

Vacuum & Autovacuum & Dead Tuple & İstatistik Analizi

- Vacuum süreçlerinin düzenli çalışıp çalışmadığı analiz edilerek tablo şişmesi (bloat) ve gereksiz disk kullanımının önüne geçilmesi hedeflenir.
- Autovacuum mekanizmasının etkinliği incelenerek otomatik bakım süreçlerinin sistem yükü ve performans üzerindeki etkisi değerlendirilir.
- Tablolarda oluşan dead tuple yoğunluğu analiz edilerek performans kaybına ve gereksiz disk tüketimine neden olan alanlar tespit edilir.
- Düzenli olarak alınan veritabanı istatistikleri sistemdeki performans değişimlerinin ve büyüme trendlerinin tarihsel olarak analiz edilmesini sağlar.

1. Vacuum & Autovacuum Analizi

Tablolarda son vacuum ve autovacuum çalışma zamanları ve çalışma sıklıkları.

```sql
SELECT
    schemaname,
    relname,
    last_vacuum,
    last_autovacuum,
    vacuum_count,
    autovacuum_count
FROM pg_stat_all_tables
ORDER BY last_autovacuum NULLS FIRST;
```

schemaname	relname	last_vacuum	last_autovacuum	vacuum_count	autovacuum_count
public	audit_log	-	-	0	0
public	temp_data	-	2025-05-10 08:15:22	0	1
public	user_sessions	2025-05-09 21:10:11	2025-05-10 06:30:12	1	5
public	orders	2025-05-10 01:05:45	2025-05-10 07:45:33	2	12
public	customers	2025-05-10 02:15:02	2025-05-10 08:05:41	1	8
sales	order_items	2025-05-09 23:40:18	2025-05-10 07:50:19	1	15
sales	invoices	2025-05-10 00:20:31	2025-05-10 08:10:25	2	10
public	products	2025-05-10 03:10:07	2025-05-10 08:12:03	1	9

2. Dead Tuple Analizi

Dead tuple sayıları yüksek olan tabloların listesi.

```sql
SELECT
    schemaname,
    relname,
    n_live_tup,
    n_dead_tup,
    last_autovacuum,
    autovacuum_count
FROM pg_stat_all_tables
ORDER BY n_dead_tup DESC
LIMIT 20;
```

schemaname	relname	n_live_tup	n_dead_tup	last_autovacuum	autovacuum_count
public	orders	1,245,678	532,452	2025-05-10 07:45:33	12
sales	order_items	5,678,234	412,987	2025-05-10 07:50:19	15
public	user_sessions	987,654	278,341	2025-05-10 06:30:12	5
public	audit_log	123,456	98,765	-	0
sales	invoices	456,789	65,432	2025-05-10 08:10:25	10
public	temp_data	234,567	45,678	2025-05-10 08:15:22	1
public	customers	678,901	32,109	2025-05-10 08:05:41	8
public	products	345,678	21,987	2025-05-10 08:12:03	9
public	categories	123,456	12,345	2025-05-10 07:20:11	4
public	product_reviews	234,567	8,765	2025-05-10 07:55:02	6

n_dead_tup değeri yüksek olan tablolar, performans ve disk kullanımı açısından risk oluşturabilir.

3. Veri Tabanı İstatistikleri Analizi

Autovacuum istatistikleri güncellese de, düzenli olarak alınan veritabanı istatistikleri sistemdeki performans değişimlerinin ve büyüme trendlerinin tarihsel olarak analiz edilmesini sağlar.

Amaç

Tüm veritabanlarında haftalık VACUUM ANALYZE çalıştırarak istatistiklerin güncel kalmasını sağlamak ve performans analizlerine sağlıklı veri kaynağı oluşturmak.

Haftalık VACUUM ANALYZE Script

```
#!/bin/sh

LOG_FILE="/var/lib/postgresql/analyze.log"

psql -d db1 -U postgres -c "VACUUM ANALYZE;" >> "$LOG_FILE" 2>&1
psql -d db2 -U postgres -c "VACUUM ANALYZE;" >> "$LOG_FILE" 2>&1

```

Açıklama

Script, db1 ve db2 veritabanlarında VACUUM ANALYZE çalıştırır ve çıktılarını /var/lib/postgresql/analyze.log dosyasına yazar.

Cron Görevi

```
# WEEKLY ANALYZE
0 6  * * 7 sh /var/lib/postgresql/scripts/weekly_analyze.sh
```

Açıklama

Cron görevi her pazar günü saat 06:00'da scripti çalıştırır.

Çalışma Akışı

Log Dosyası Örneği (analyze.log)

```
2025-05-10 06:00:01   INFO:  vacuuming "public.tablo1"
2025-05-10 06:00:02   INFO:  analyzed "public.tablo1"
2025-05-10 06:00:02   INFO:  VACUUM
2025-05-10 06:00:03   INFO:  ANALYZE
2025-05-10 06:00:04   INFO:  vacuuming "public.tablo2"
2025-05-10 06:00:05   INFO:  analyzed "public.tablo2"
2025-05-10 06:00:05   INFO:  VACUUM
2025-05-10 06:00:06   INFO:  ANALYZE
```

Faydaları

- İstatistikler güncel kalır
- Sorgu performansı artar
- Performans trendleri izlenebilir
- Olası sorunlar erken tespit edilir

Transaction ID Wraparound

PostgreSQL'de Transaction ID (XID) 2 milyar sınırına yaklaştığında veritabanı kendini korumak için read-only moda geçebilir. Bu nedenle XID yaşını düzenli olarak izlemek en kritik healthcheck adımıdır.

SORUN:
XID age değeri > 1.8B ise **ACİL** aksiyon alınmalıdır.
VACUUM FREEZE çalıştırın, gerekirse uygulamayı durdurun.

1. Veritabanı Bazında XID Yaşı

```sql
SELECT
    datname,
    age(datfrozenxid) AS xid_age
FROM pg_database
ORDER BY age(datfrozenxid) DESC;
```

Her veritabanı için en son frozen transaction ID'ye göre mevcut XID yaşını gösterir.

2. Tablo Bazında XID Yaşı (En Riskli 30 Tablo)

```sql
SELECT
    n.nspname AS schema_name,
    c.relname AS table_name,
    age(c.relfrozenxid) AS xid_age,
    pg_size_pretty(pg_total_relation_size(c.oid)) AS size
FROM pg_class c
JOIN pg_namespace n ON n.oid = c.relnamespace
WHERE c.relkind IN ('r','m','t')
ORDER BY age(c.relfrozenxid) DESC
LIMIT 30;
```

XID yaşı en yüksek olan (wraparound riski en fazla) tabloları gösterir.
r: normal tablo, m: materialized view, t: TOAST tablo

3. XID Age Eşik Tablosu

XID Age	Yaklaşık Değer	Durum	Açıklama	Önerilen Aksiyon
< 500 milyon	< 500M	GÜVENLİ	Güvenli seviyede.	Rutin izlemeye devam edin.
500M – 1B	500M – 1.000M	TAKİP ET	Yakından takip edilmelidir.	Autovacuum ayarlarını kontrol edin.
1B – 1.5B	1.000M – 1.500M	RİSKLİ	Autovacuum agresif devreye girer.	Autovacuum'un düzgün çalıştığından emin olun.
> 1.8B	1.800M – 2.000M	ÇOK YÜKSEK RİSK	PostgreSQL log atar, read-only'e geçer.	**ACİL: VACUUM FREEZE** çalıştırın, gerekirse uygulamayı durdurun.
2.1B	2.100M+	KRİTİK (WRAPAROUND)	Veri erişilemez hale gelir.	**ACİL: VACUUM FREEZE** ve gerekli müdahaleyi yapın (veri kaybı riski yüksektir).

İpucu

autovacuum_freeze_max_age = 150000000 (150M) yaparak daha sık freeze sağlayabilirsiniz.

Multixact ID Wraparound

Row-level lock işlemleri için kullanılan Multixact ID'ler de wraparound riski taşır. Aşağıdaki sorgu ile takip edebilirsiniz:

```sql
mxid_age(datminmxid)
```

Faydaları | İstatistikler güncel kalır | Sorgu performansı artar | Performans trendleri izlenebilir | Olası sorunlar erken tespit edilir

Bağlantı Durumu Dağılımı

Ne kontrol edilir?

- Bağlantı state dağılımı
- Uzun idle session'lar
- idle in transaction oturumlar

Neden önemli?

- Lock riski
- Vacuum gecikmesi
- Bloat artışı
- Uygulama transaction hataları

1) Bağlantıların state dağılımı

```sql
SELECT state, count(*)
FROM pg_stat_activity
GROUP BY state
ORDER BY count DESC;
```

state	count
idle	86
active	12
idle in transaction	3
idle in transaction (aborted)	1
fastpath function call	0

Sağlıklı: Çoğunluk idle, az sayıda active. idle in transaction < 5

2) 10 dakikadan uzun idle kalan bağlantılar

```sql
SELECT pid, usename, application_name, state,
       now() - state_change AS duration
FROM pg_stat_activity
WHERE state != 'active'
  AND state_change < now() - interval '10 minutes'
ORDER BY duration DESC;
```

pid	usename	application_name	state	duration
24567	app_user	my_app_backend	idle	00:37:42
24501	report_user	reporting_service	idle	00:28:15
24410	etl_user	etl_job	idle	00:19:03
24322	monitor	pgadmin4	idle	00:15:47
24111	app_user	my_app_backend	idle	00:12:33

10 dakikadan uzun süre idle durumda olan bağlantılar listelenir.

3) En uzun idle in transaction oturumlar

```sql
SELECT pid, usename, application_name, state,
       now() - xact_start AS xact_duration,
       left(query, 100) AS last_query
FROM pg_stat_activity
WHERE state = 'idle in transaction'
ORDER BY xact_duration DESC NULLS LAST;
```

pid	usename	application_name	state	xact_duration	last_query
24345	app_user	my_app_backend	idle in transaction	01:42:17	UPDATE orders SET status = '...'
23111	app_user	my_app_backend	idle in transaction	00:55:03	INSERT INTO audit_log (use...
22999	report_user	reporting_service	idle in transaction	00:33:28	SELECT * FROM big_table ...
22888	batch_user	batch_job	idle in transaction	00:15:44	DELETE FROM temp_data ...

idle in transaction birikmesi sorun yaratır: Lock'a yol açar, vacuum'u engeller.

İPUÇLARI: ZAMAN AŞIMI (TIMEOUT) AYARLARI

idle_in_transaction_session_timeout = 300000 (5 dk)

Belirtilen süreden uzun süredir idle durumda olan transaction'ları otomatik sonlandırır.
Lock birikmesini ve vacuum engelerini önler.

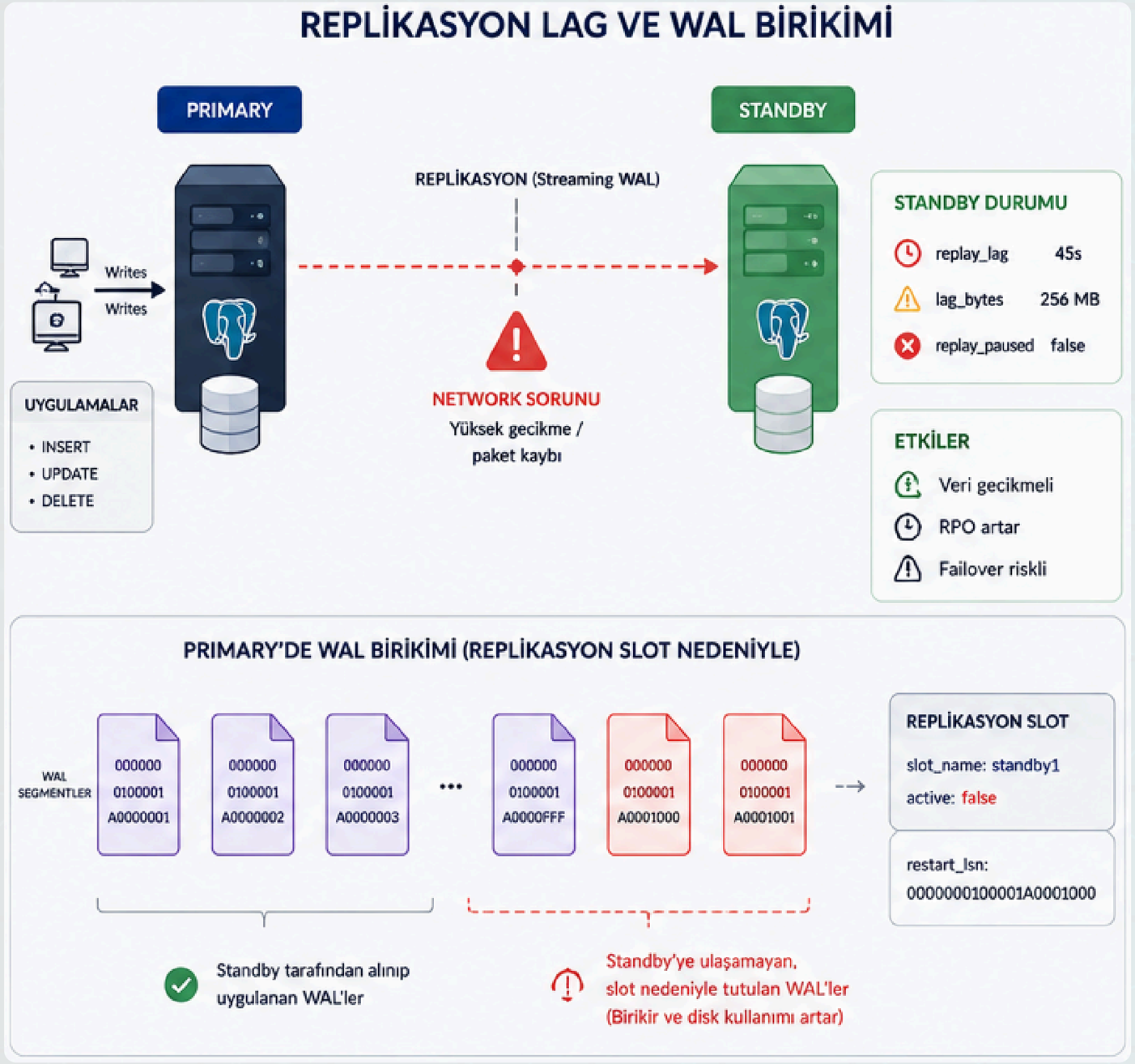
```
# 5 dakika, (300000 ms) sonrası
idle in transaction oturumlarını sonlandırır
idle_in_transaction_session_timeout = 300000
```

statement_timeout = 60000 (1 dk)

Belirtilen süreden uzun çalışan sorguları otomatik olarak durdurur.
Runaway (kontrolden çıkan) sorguların önüne geçer.

```
# 1 dakika (60000 ms) sonrası
uzun süren sorguları iptal et
statement_timeout = 60000
```

Uygulama ihtiyaçlarınıza göre süreleri ayarlayın, Çok agresif timeout'lar meşru uzun işlemleri de etkileyebilir.



Replikasyon Kontrolleri

Tipik nedenler:

- Standby kapalı
- Network bağlantısı kopuk
- primary_conninfo hatalı
- primary_slot_name yanlış
- Replication user / pg_hba problemi
- Logical consumer veya subscription durmuş

İPUÇLARI

- Lag > 100MB veya replay_lag > 30s ise müdahale edin.
- Sorgu sonuçları boşsa standby bağli değil (replikasyon kopuk).
- Gereksiz slot'ları temizleyin.
- Network, CPU, I/O ve WAL gönderen bağlantıyı kontrol edin.

SONUÇ

Network problemi nedeniyle standby WAL'leri zamanında alamıyor.

Replikasyon slot (standby1) aktif olmadığı için WAL'ler silinmiyor ve birikiyor.

WAL birikimi → disk dolabilir → primary yazma engellenir!

2.1 REPLİKASYON LAG (PRIMARY'DE ÇALIŞTIR)

Ne yapıyorsunuz: Standby sunucuların ne kadar geride kaldığını ölçersiniz.

```sql
SELECT
  client_addr,
  application_name,
  state,
  sent_lsn,
  write_lsn,
  flush_lsn,
  replay_lsn,
  (sent_lsn - replay_lsn) AS replication_lag_bytes,
  pg_size_pretty(sent_lsn - replay_lsn) AS lag_pretty,
  write_lag,
  flush_lag,
  replay_lag,
  sync_state
FROM pg_stat_replication
ORDER BY replication_lag_bytes DESC;
```

client_addr	application_name	state	sent_lsn	write_lsn	flush_lsn	replay_lsn	replication_l	ag_pretty	write_lag	flush_lag	replay_lag	sync_
10.0.0.12	standby-2	streaming	0/7000100	0/70000F0	0/70000E0	0/6FFF000	1,048,576	256 MB	00:00:00.120	00:00:00.110	00:00:00.450	asy
10.0.0.11	standby-1	streaming	0/7000100	0/70000F8	0/70000F0	0/6FFF820	65,536	15 MB	00:00:00.005	00:00:00.004	00:00:00.020	sy
10.0.0.13	standby-3	streaming	0/7000100	0/70000F0	0/70000E8	0/6FFFEC0	2,097,152	2 MB	00:00:00.000	00:00:00.000	00:00:00.000	asy
10.0.0.14	standby-4	streaming	0/7000100	0/70000F0	0/70000E0	0/70000CC0	524,288	512 kB	00:00:00.000	00:00:00.000	00:00:00.000	pote

- **Beklenen:** replay_lag < 1 saniye, lag_bytes < 50MB
- **Sağlıklı:** Lag MB düzeyinde, saniyeler içinde kapanıyor. `sync_state = 'async'` normal, 'sync' ise synchronous replikasyon aktif demektir.
- **Sorun:** Lag > 100MB veya replay_lag > 30s → failover durumunda veri kaybı riski.
- **İpucu:** Sorgu boş dönüyorsa standby bağlı değil, replikasyon kopuk demektir. `sync_state = 'potential'` ise o standby yedek sync sunucu anlamına gelir.

2.2 STANDBY DURUMU (STANDBY'DE ÇALIŞTIR)

Ne yapıyorsunuz: Standby'in recovery modunda olduğunu ve WAL alıp uyguladığını doğrularsınız.

```sql
SELECT pg_is_in_recovery() AS is_standby;
```

is_standby
t
(true)

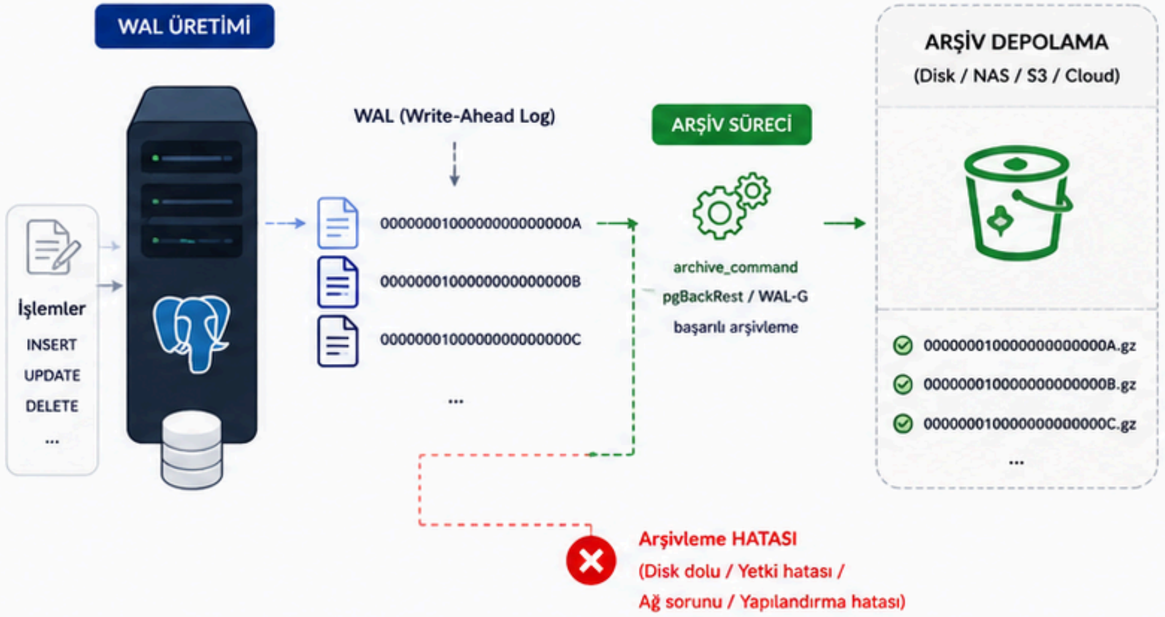
```sql
SELECT
  pg_last_wal_receive_lsn() AS received,
  pg_last_wal_replay_lsn() AS replayed,
  pg_last_xact_replay_timestamp() AS last_replay_time,
  now() - pg_last_xact_replay_timestamp() AS replay_delay,
  pg_is_wal_replay_paused() AS replay_paused;
```

received	replayed	last_replay_time	replay_delay	replay_paused
0/70000D8	0/70000D8	2025-05-10 08:15:22+03	00:00:03	f (false)

- **Beklenen:** `is_standby = true`, `replay_delay < 30s`, `replay_paused = false`
- **Sağlıklı:** `received` ve `replayed` LSN değerleri sürekli artıyor.
- **Sorun:** `received = replayed` ama delay artıyorsa WAL gelmiyor → ağ veya primary sorunu.
- `replay_paused = true` ise biri elle `pg_wal_replay_pause()` çağırmış; `pg_wal_replay_resume()` ile devam ettir.
- **İpucu:** Patroni kullanıyorsan `patronictl -c patroni.yml list` ile cluster durumunu kontr `pg_promote()` fonksiyonu ile standby'i primary yapabilirsin (Patroni varsa bunu Patroni üzerinden yapın).

2.3 WAL ARŞİV DURUMU

WAL dosyalarının başarıyla arşivlenip arşivlenmediğini kontrol edin.
Bu PITR (Point-in-Time Recovery) için zorunludur.



WAL arşivleri olmadan PITR yapılamaz. Arşivleme sürekliliği veri kurtarılabilirliği için kritiktir.

SQL SORGUSU

```sql
SELECT
  archived_count,
  last_archived_wal,
  last_archived_time,
  failed_count,
  last_failed_wal,
  last_failed_time,
  now() - last_archived_time AS since_last_archive,
  round(archived_count::float / nullif(archived_count + failed_count, 0) * 100, 2) AS success_pct
FROM pg_stat_archiver;
```

ÖRNEK SONUÇ

archived_count	last_archived_wal	last_archived_time	failed_count	last_failed_wal	last_failed_time	since_last_archive	success_pct
15238	000000010000001B000000FF	2025-05-10 08:12:45.123+03	0	-	-	00:01:12.345	100.00

BEKLENEN

- failed_count = 0
- son arşiv < 5 dakika önce

SAĞLIKLI

- failed_count sıfır,
  düzenli arşiv devam ediyor.

SORUN

- failed_count artıyorsa
  pgBackRest/WAL-G yapılandırması
  bozuk veya hedef depolama alanı dolu.

İPUÇLARI

.ready dosyaları birikmişse arşiv tıkalı demektir.

```
$ ls -l pg_wal/archive_status/
-rw------- 1 postgres postgres 0 May 10 08:10 000000010000000000000A.ready
-rw------- 1 postgres postgres 0 May 10 08:11 000000010000000000000B.ready
...
```

last_failed_wal hangi dosyada takıldığını gösterir.
O dosyayı elle arşivlemeyi deneyin.

```
$ pgBackRest --stanza=main archive-push 000000010000000000000A
# veya
$ wal-g wal-push 000000010000000000000A
```

Arşiv hedefinize yazılabildiğinden emin olun.

- [x] Disk alanı yeterli mi?
- [x] Ağ bağlantısı sağlıklı mı?
- [x] Yetkiler doğru mu?

NOT

Arşivleme kesintisiz çalışmalıdır.
Arşivde boşluk oluşursa,
PITR sırasında veri kaybı yaşanabilir.
Düzenli olarak kontrol edin!

Uzun Çalışan Sorgular / En Yavaş Sorgular

Uzun süren sorgular sistem kaynaklarını tüketir, diğer işlemleri bekletir ve uygulama performansını olumsuz etkiler.
Aşağidaki sorgu ile aktif ve 30 saniyeden uzun süren sorgular tespit edilir, pgBadger raporları ise yavaş sorguların geçmişe dönük analizini sağlar.

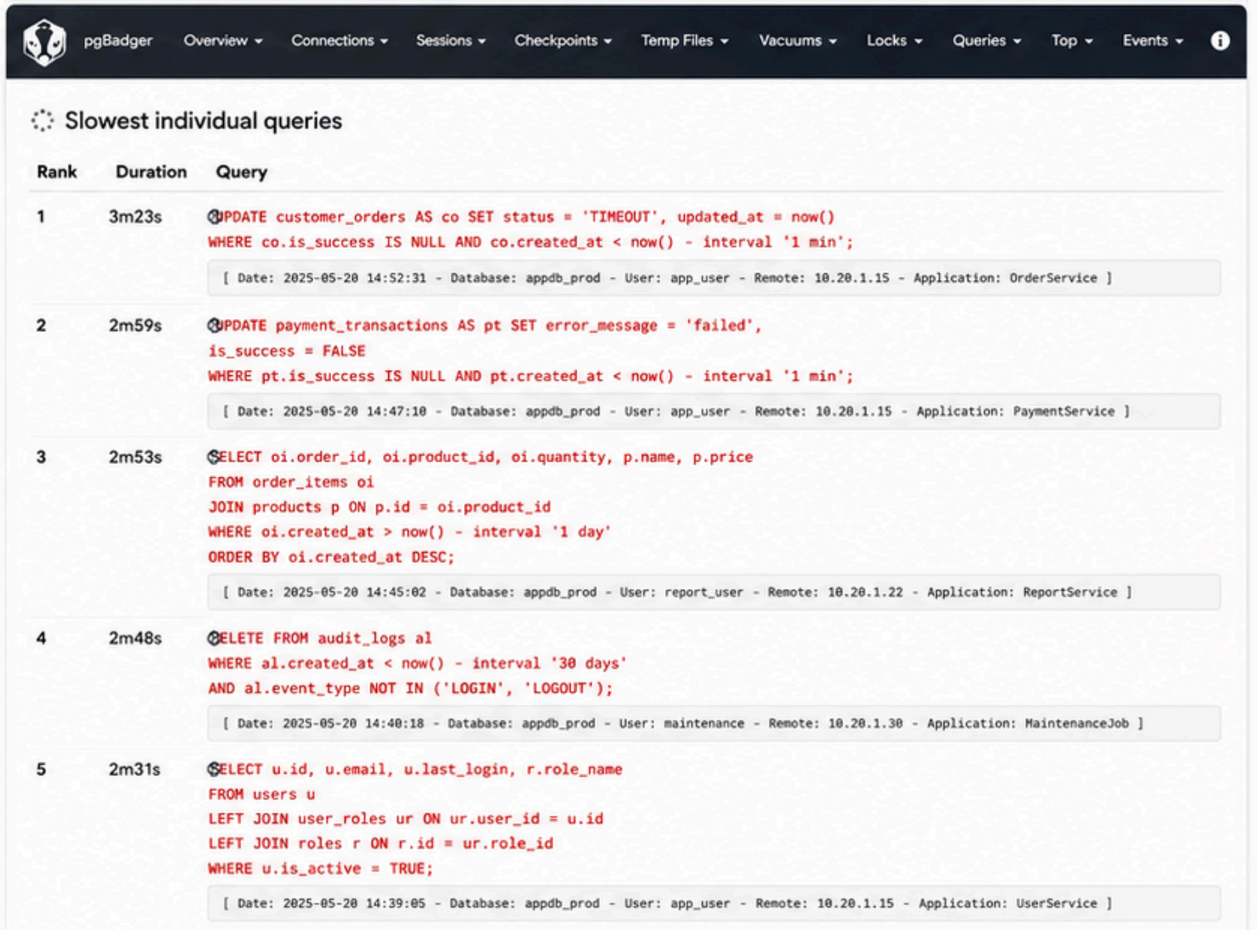
1. Aktif ve Uzun Süreli Sorgular (30 saniyeden uzun)

```sql
SELECT
    pid,
    now() - query_start AS duration,
    state,
    wait_event_type,
    wait_event,
    left(query, 120) AS query_snippet,
    usename,
    application_name,
    client_addr
FROM pg_stat_activity
WHERE state = 'active'
  AND query_start < now() - interval '30 seconds'
  AND query NOT LIKE 'autovacuum%'
ORDER BY duration DESC;
```

pid	duration	state	wait_event_type	wait_event	query_snippet	usename	application_name	client_addr
84215	00:04:52	active	IO	DataFileRead	SELECT o.id, o.customer_id, o.amount, o.status, ...	app_user	OrderService	10.20.1.15
84102	00:02:17	active	Lock	transactionid	UPDATE orders SET status = 'SHIPPED' WHERE id IN (SELECT id FROM ...	app_user	OrderService	10.20.1.15
83988	00:01:36	active	Client	ClientRead	INSERT INTO order_items (order_id, product_id, qty, price) VALUES ...	app_user	OrderService	10.20.1.15
83877	00:01:02	active	IO	DataFileRead	SELECT p.id, p.name, p.category_id, p.price FROM products p ...	report_user	ReportService	10.20.1.22
83731	00:00:45	active	IO	DataFileRead	SELECT * FROM inventory WHERE quantity < 10 ORDER BY ...	app_user	InventoryService	10.20.1.28

30 saniyeden uzun süren aktif sorgular listelenir.
Sorgu süresi, bekleme olayları ve istemci bilgileri ile performans sorunlarının kaynağı tespit edilir.

2. pgBadger Raporu – En Yavaş Sorgular (Örnek Kesit)



pgBadger log analizi ile en yavaş sorguların geçmişe dönük sürelerini, sıklıklarını ve trendlerini raporlar.
Geçmişe dönük analiz, performans sorunlarının kök nedenini bulmada kritik öneme sahiptir.

Kilitleme (Lock) Analizi

Hangi sorgunun hangisini bloke ettiğini bulur, lock türlerini ve hangi tablolarda beklemeler olduğunu analiz eder.
Lock problemleri uygulama katmanında yavaşlama, timeout ve kaynak tüketimine neden olabilir.

1. Bloke Olan ve Bloke Eden Sorgular

```sql
SELECT
    blocked.pid AS blocked_pid,
    blocked.username,
    blocked.application_name,
    blocking.pid AS blocking_pid,
    blocking.username AS blocking_user,
    left(blocked.query, 100) AS blocked_query,
    left(blocking.query, 100) AS blocking_query,
    now() - blocked.query_start AS blocked_duration,
    now() - blocking.query_start AS blocker_running_since
FROM pg_stat_activity blocked
JOIN pg_stat_activity blocking
  ON blocking.pid =  ANY(pg_blocking_pids(blocked.pid))
WHERE cardinality(pg_blocking_pids(blocked.pid)) > 0
ORDER BY blocked_duration DESC;
```

Hangi sorgunun hangi sorguyu bloke ettiğini ve ne kadar süredir beklediğini gösterir.

2. Lock Türleri ve Hangi Tablolarda Bekleme Var

```sql
SELECT
    l.pid, l.mode, l.granted,
    c.relname AS table_name,
    l.locktype
FROM pg_locks l
LEFT JOIN pg_class c ON l.relation = c.oid
WHERE NOT l.granted
ORDER BY l.pid;
```

Grant edilmemiş (beklemedeki) lock'ları, lock türleriyle birlikte hangi tablolarda olduğunu gösterir.

Beklenen / Sağlıklı Sonuç

Hiç satır yok = lock yok.

blocked_pid	blocking_pid	blocked_duration	blocker_running_since
(0 rows)			

pid	mode	granted	table_name	locktype
(0 rows)				

Sorunlu Durum

Uzun lock zinciri → uygulama katmanında transaction yönetimi bozuk olabilir.
AccessExclusiveLock en tehlikelisidir:
ALTER TABLE, VACUUM FULL, TRUNCATE gibi işlemler tutar.

blocked_pid	blocking_pid	blocked_duration
24567	12345	00:12:47
24601	12345	00:09:32
24622	12345	00:07:18
...	...	...

İpucu

Bloke eden pid'i bulduktan sonra

```sql
SELECT pg_terminate_backend(<blocking_pid>);
```

ile öldürebilirsin. Önce uygulamayı bildir.

Uygulamanın lock için sonsuza kadar beklemesini önlemek için

```sql
SET lock_timeout = '5000';  -- 5 saniye
```

Faydaları

- Lock problemlerini hızlı tespit etmeyi sağlar
- Bekleme sürelerini azaltır ve performansı iyileştirir
- Uygulamada timeout ve hata riskini minimize eder
- Kaynak tüketimini kontrol altında tutmaya yardımcı olur
- Kullanıcı deneyimini iyileştirir

Cache Hit Oranı

PostgreSQL'in veriyi RAM'den mi yoksa diske mi okuduğunu ölçer.
Yüksek cache hit oranı, düşük disk I/O ve yüksek performans anlamına gelir.

Hedef

OLTP sistemlerde cache hit oranı `%99+` olmalıdır.

1. Genel Veritabanı Düzeyi Hit Oranı

```sql
SELECT
    datname,
    blks_read,
    blks_hit,
    round(blks_hit::numeric / nullif(blks_hit + blks_read, 0) * 100, 2) AS hit_pct
FROM pg_stat_database
WHERE dataname NOT IN ('template0', 'template1')
ORDER BY blks_read DESC;
```

Veritabanı genelinde okuma işlemlerinin ne kadarının bellekten yapıldığını gösterir.
Hit oranı ne kadar yüksekse disk baskısı o kadar düşüktür.

2. Tablo Cache Hit (En Çok Diskten Okunan Tablolar)

```sql
SELECT
    schemaname, tablename,
    heap_blks_read,
    heap_blks_hit,
    round(heap_blks_hit::numeric /
        nullif(heap_blks_hit + heap_blks_read, 0) * 100, 2) AS hit_pct,
    pg_size_pretty(heap_blks_read * 8192) AS disk_read_volume
FROM pg_statio_user_tables
WHERE heap_blks_read + heap_blks_hit > 0
ORDER BY heap_blks_read DESC LIMIT 20;
```

Tablo bazında diskten okunan blok sayısını, hit oranını ve yaklaşık disk okuma hacmini gösterir.
En fazla disk I/O üreten tabloları tespit etmeye yardımcı olur.

Beklenen / Sağlıklı

- %99+ cache hit → RAM yeterli, disk baskısı yok.
- Sistem performansı optimal seviyededir.

Sorunlu Durum

- %95'in altında → shared_buffers veya toplam RAM yetersiz olabilir.
- Disk I/O artar, sorgular yavaşlar.
- Hangi tablonun en fazla disk I/O ürettiğine bak.

İpuçları

- `shared_buffers` = RAM'in %25'i başlangıç noktasıdır.
- `effective_cache_size` = RAM'in %75'i başlangıç noktasıdır.
- Büyük sunucularda `shared_buffers`'ı `8-16 GB`'a kadar artırabilirsin.
- İstatistikler sunucu başlangıcından bu yana kümülatiftir.
- İstatistikleri sıfırlamak için dikkatli şekilde `SELECT pg_stat_reset();` komutu kullanılabilir.

Neden Önemli?

Cache hit oranı düşükse disk I/O artar, CPU beklemeleri yükselir ve genel sistem performansı olumsuz etkilenir.

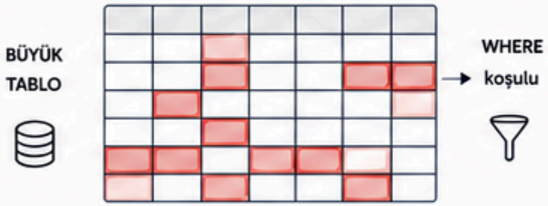
SEQUENTIAL SCAN TESPİTİ (EKSİK INDEX İŞARETİ)

Hangi tabloların tam tarama (seq scan) yaptığını tespit ederek,
index ile yapılması gereken taramaları belirleriz.

Temel Mantık: Büyük tablolarda seq scan, gereksiz I/O ve CPU tüketir. Doğru index ve güncel istatistikler ile performans ciddi şekilde iyileşir.

SEQUENTIAL SCAN (TAM TARAMA)

Tablonun tamamını okur.

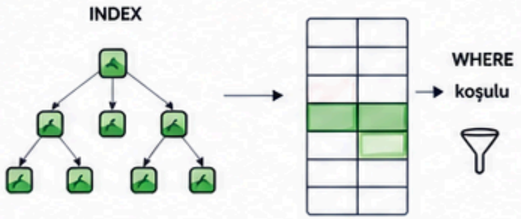


- Tüm satırlar okunur, sonra filtrelenir.
- Yavaş ve pahalıdır.

Büyük tablolarda gereksiz seq scan performansı ciddi şekilde düşürür.

INDEX SCAN (INDEX TARAMA)

Sadece ilgili satırlar okunur.



- Index üzerinden hızlıca bulunur.
- Sadece gerekli satırlar okunur.

Doğru index ile sorgular çok daha hızlı çalışır, CPU, I/O ve disk kullanımı azalır.

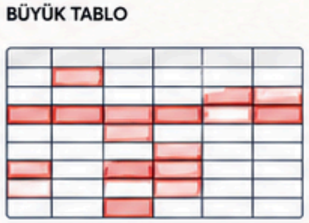
NE ZAMAN NORMALDİR?

- Küçük tablolar (< 1000 satır)
- Sorgulanan satır oranı yüksekse
- Planner doğru karar verir

Seq scan tek başına problem değildir. Doğru index ve güncel istatistiklerle performans iyileşir.

NE ZAMAN SORUNLUDUR?

- Büyük tabloda sürekli seq scan
- WHERE koşulunda index yok
- İstatistikler bayat



Büyük tablolarda gereksiz seq scan zaman ve kaynak israfına yol açar.

NASIL İNCELEMELİ VE ÇÖZMELİ?

1. **TESPİT ET**
   Hangi tablolarda seq scan oranı yüksek?
2. **PLAN İNCELE**
   EXPLAIN (ANALYZE, BUFFERS) ile sorgu planını incele
3. **İSTATİSTİK GÜNCELLE**
   ANALYZE tablo_adi; ile istatistikleri yenile
4. **INDEX EKLE**
   CREATE INDEX CONCURRENTLY ile index oluştur (prod'da kilitlemez)
5. **TEKRAR KONTROL ET**
   Seq scan oranı düştü mü kontrol et

EXPLAIN (ANALYZE, BUFFERS) ÖRNEĞİ

SEQ SCAN (KÖTÜ ÖRNEK)

```
Seq Scan on orders  (cost=0.00..125678.32 rows=846211 width=512)
  (actual time=1256.123..1456.789 rows=5 loops=1)
  Filter: (customer_id = 12345)
  Rows Removed by Filter: 8542112
  Buffers: shared hit=125678 read=45233
  Planning Time: 0.345 ms
  Execution Time: 1456.845 ms
```

INDEX SCAN (İYİ ÖRNEK)

```
Index Scan using idx_orders_customer_id on orders
  (cost=0.43..8.45 rows=10 width=552)
  (actual time=0.123..0.456 rows=5 loops=1)
  Index Cond: (customer_id = 12345)
  Buffers: shared hit=8 read=1
  Planning Time: 0.210 ms
  Execution Time: 0.512 ms
```

YARARLARI

- Sorgu performansı artar
- Disk ve CPU kullanımı azalır
- Index kullanım oranı artar
- Kullanıcı deneyimi iyileşir
- Kaynak israfı önlenir

İLGİLİ ARAÇ VE KOMUTLAR

İstatistikleri güncelle	`ANALYZE tablo_adi;`
Planı incele	`EXPLAIN (ANALYZE, BUFFERS) <sorgu>;`
Index oluştur (kilitlemez)	`CREATE INDEX CONCURRENTLY idx_adi ON tablo (kolon);`
İstatistikleri sıfırla (opsiyonel)	`SELECT pg_stat_reset();`

ÖZET: Büyük tablolarda gereksiz sequential scan tespit edilip, doğru index ve güncel istatistiklerle performans önemli ölçüde iyileştirilebilir.

KULLANILMAYAN INDEX'LER

Hiç veya nadiren kullanılan, sadece disk kaplayan ve her INSERT/UPDATE/DELETE'te maliyet yaratan indexleri tespit edersin.

NASIL ÇALIŞIR?

pg_stat_user_indexes görünümündeki idx_scan değeri, bir indexin kaç kez kullanıldığını gösterir.

idx_scan = 0 olanlar hiç kullanılmamıştır.

Çok düşük olanlar (< 100) ise nadiren kullanılmaktadır.

SORUN NEDİR?

- Her yazma işleminde (INSERT/UPDATE/DELETE) gereksiz yük oluşturur.
- Disk alanı tüketir.
- VACUUM süresini uzatır.

TEMİZLİK ADIMLARI

1. Önce indexi devre dışı bırakmak için yeniden adlandır.

```
ALTER INDEX index_adi RENAME TO index_adi_DISABLED_20240601;
```

2. Uygulamanı test et ve bir süre (ör. 1–2 hafta) gözlemle.

3. Sorun yoksa indexi sil.

```
DROP INDEX CONCURRENTLY index_adi_DISABLED_20240601;
```

İPUCU

PRIMARY KEY ve UNIQUE constraint'lerin oluşturduğu indexleri silme. `pg_index.indisprimary = true` veya `indisunique = true` olanlar constraint'tir.

1) HİÇ KULLANILMAYAN INDEX'LER (idx_scan = 0)

```
SELECT
   schemaname,
   relname AS tablename,
   indexrelname AS indexname,
   pg_size_pretty(pg_relation_size(indexrelid)) AS index_size,
   idx_scan,
   idx_tup_read,
   idx_tup_fetch
FROM pg_stat_user_indexes
WHERE idx_scan = 0
 AND schemaname NOT IN ('pg_catalog','information_schema')
ORDER BY pg_relation_size(indexrelid) DESC;
```

ÖRNEK SONUÇ

schemaname	tableneme	indexname	index_size	idx_scan	idx_tup_read	idx_tup_fetch
public	orders	idx_orders_old_status	512 MB	0	0	0
public	customers	idx_customers_temp	256 MB	0	0	0
public	events	idx_events_unused	128 MB	0	0	0
public	logs	idx_logs_debug	64 MB	0	0	0
...	...	...	...	...	...	...

BEKLENEN: Tüm indexler kullanılmalıdır.

2) NADİREN KULLANILAN INDEX'LER (idx_scan < 100, büyük tablo)

```
SELECT
   s.schemaname,
   s.relname AS tablename,
   s.indexrelname AS indexname,
   pg_size_pretty(pg_relation_size(s.indexrelid)) AS index_size,
   s.idx_scan,
   t.n_live_tup AS row_count,
   pg_size_pretty(pg_total_relation_size(t.relid)) AS table_size
FROM pg_stat_user_indexes s
JOIN pg_stat_user_tables t ON s.relid = t.relid
WHERE s.idx_scan < 100
 AND t.n_live_tup > 50000
 AND s.schemaname NOT IN ('pg_catalog','information_schema')
ORDER BY pg_relation_size(s.indexrelid) DESC
LIMIT 20;
```

ÖRNEK SONUÇ

schemaname	tableneme	indexname	index_size	idx_scan	row_count	table_size
public	orders	idx_orders_customer_old	1.2 GB	45	12,458,321	8.7 GB
public	payments	idx_payments_method	768 MB	32	2,154,987	3.1 GB
public	events	idx_events_type_old	512 MB	18	1,102,345	2.4 GB
public	sessions	idx_sessions_expired	256 MB	7	987,654	1.8 GB
...	...	...	...	...	...	...

İPUCU: Bu indexler periyodik job veya nadir raporlarda kullanılabilir. Silmeden önce yeterli süre gözlemle.

NOT: Bu istatistikler sunucu restart veya `pg_stat_reset()` ile sıfırlanır.

Bir indexi silmeden önce en az 2–4 hafta istatistik topla. Deployment veya batch job'lar nedeniyle bazı indexler periyodik kullanılabilir.

INDEX BLOAT ANALİZİ

Indexlerin içindeki ölü sayflanımı tespit ederek şişen (bloat) indexleri belirle.
Tablolar gibi indexler de zamanla şişer ve REINDEX gerektirebilir.

NE YAPIYORSUN?

Indexlerin boyutunu ve kullanımını inceleyerek bloat şüphesi olan indexleri tespit ediyorsun. Gerekirse pgstattuple ile ayrıntılı bloat ölçümü yapıp REINDEX ile temizliyorsun.

BLOAT NEDİR?

- DELETE ve UPDATE işlemleri index içinde ölü (dead) girdiler bırakır.
- B-tree indexleri zamanla şişer, disk alanı ve I/O maliyeti artar.
- Otomatik VACUUM sadece heap'i temizler, indexleri değil.

BLOAT GÖSTERGELERİ

- Index boyutu tablonun büyümesiyle orantısız artıyorsa
- idx_scan düşük veya değişmiyorsa
- pgstattuple'da avg_leaf_density < 50% ise

İPUCU

İki farklı zamanda (ör. arayla 2 gün/hafta) aynı sorguyu çalıştır.
Boyut artıyor ama idx_scan artmıyorsa bloat şüphesi yüksektir.
pgstattuple büyük indexlerde yavaş çalışır, dikkatli kullan.

3) DETAYLI BLOAT ÖLÇÜMÜ (PGSTATINDEX)

pgstattuple extension gerekir.

```
CREATE EXTENSION IF NOT EXISTS pgstattuple;
```

```
SELECT * FROM pgstatindex('index_adi');
```

1) BÜYÜK INDEXLERİ LİSTELE (BLOAT ŞÜPHESİ İÇİN)

```
SELECT
    schemaname,
    relname AS tablename,
    indexrelname AS indexname,
    pg_size_pretty(pg_relation_size(indexrelid)) AS index_size,
    idx_scan,
    idx_tup_read,
    idx_tup_fetch
FROM pg_stat_user_indexes
WHERE pg_relation_size(indexrelid) > 10 * 1024 * 1024  -- 10MB üzeri
  AND schemaname NOT IN ('pg_catalog')
ORDER BY pg_relation_size(indexrelid) DESC
LIMIT 20;
```

ÖRNEK SONUÇ

schemaname	tablename	indexname	index_size	idx_scan	idx_tup_read	idx_tup_fetch
public	orders	idx_orders_created_at	1.8 GB	42	15,214,567	3,245,678
public	payments	idx_payments_method	768 MB	32	2,154,987	1,254,321
public	events	idx_events_type	512 MB	18	1,102,345	987,654
public	sessions	idx_sessions_user_id	256 MB	7	987,654	543,210
...	...	...	...	...	...	...

2) AYNI INDEXİN BÜYÜMESİNİ ZAMANLA TAKİP ET

```
SELECT
    indexrelname,
    pg_size_pretty(pg_relation_size(indexrelid)) AS now_size,
    idx_scan
FROM pg_stat_user_indexes
WHERE relname = 'tablo_adi'
ORDER BY pg_relation_size(indexrelid) DESC;
```

NASIL KULLANILIR?

Bu sorguyu iki farklı zamanda (ör. arayla 2 gün) çalıştır.
Index boyutu artıyor ama idx_scan artmıyorsa → bloat şüphesi yüksektir.

ÖRNEK SONUÇ

indexrelname	now_size	idx_scan
idx_orders_created_at	1.8 GB	42
idx_orders_created_at	2.1 GB (2 gün sonra)	43
idx_orders_created_at	2.5 GB (1 hafta sonra)	45
...	...	...

ÖRNEK ÇIKTI

metrik	değer
avg_leaf_density	28.45
leaf_fragmentation	62.31
deleted_pages	15432
live_pages	10250

DEĞERLENDİRME

avg_leaf_density < 50% ise ciddi bloat vardır. Indexin yarısı boşa gitmiş demektir.

ÇÖZÜM

Şişmiş indexleri yeniden oluştur.

```
-- Tek index için (kilitsiz - PG12+)
REINDEX INDEX CONCURRENTLY index_adi;

-- Tablodaki tüm indexler için (kilitsiz - PG12+)
REINDEX TABLE CONCURRENTLY tablo_adi;

-- Alternatif: pg_repack extension'ı
-- kilitsiz VACUUM FULL + REINDEX yapabilir.
```

NOTLAR

- B-tree indexleri zamanla şişer, REINDEX şarttır.
- autovacuum sadece heap'i temizler.
- autovacuum_vacuum_scale_factor'ı düşürmek heap bloat'ı azaltır, index bloat için REINDEX gerekir.
- Kritik olan indexlerde düzenli kontrol performans ve disk tasarrufu sağlar.

ÖNERİ

1. **Tespit et** — Büyük indexleri bul
2. **İzle** — Boyut ve kullanım değişimini takip et
3. **Ölç** — pgstattuple ile bloat'ı ölç
4. **Temizle** — REINDEX ile şişmiş indexleri yeniden oluştur.

BİZİ DİNLEDİĞİNİZ İÇİN

TEŞEKKÜRLER :)