PostgreSQL 18'de Gelen Yenilikler

Şahap Aşçı

Cooksoft - sahap.asci@cooksoft.com.tr

@sahapasci

sahapasci

Asynchronous I/O

- `sync` → Geleneksel senkron davranış (PG17)
- `worker` → PostgreSQL'in arka planda I/O işçileri kullanması
- `io_uring` → Linux 5.1+
    a. Linux çekirdeğinde yüksek performanslı asenkron giriş/çıkış (I/O) işlemleri için bir API'dir. 2019'da tanıtıldı.

Asynchronous I/O

PostgreSQL 18 Asenkron I/O Karşılaştırması

- Sunucu kaynak bilgisi : 4vCPUs 8GiB RAM
- Test Konfigürasyonu
  a. Clients : 32
  b. Threads : 8
  c. Duration : 300s
  d. 80GB veri seti

Asynchronous I/O

Test Sonuçları (ASYNC_OFF)

Test Türü	TPS	Ortalama Gecikme (ms)
Read-Write	2130	15.023
Read-Only	3979	8.041
Write	2007	15.946

postgresql.auto.conf içeriği :

effective_cache_size = '6GB'
max_worker_processes = '8'
random_page_cost = '1.1'
shared_buffers = '2GB'
io_method = 'sync'

Asynchronous I/O

Test Sonuçları (ASYNC_ON)

Test Türü	TPS	Ortalama Gecikme (ms)
Read-Write	2198	14.558
Read-Only	3901	8.202
Write	2090	15.313

postgresql.auto.conf içeriği :

effective_cache_size = '6GB'
max_worker_processes = '8'
random_page_cost = '1.1'
shared_buffers = '2GB'
io_method = 'worker'
io_workers = '3'
effective_io_concurrency = '16'

Asynchronous I/O

TPS Karşılaştırması ve Fark (%)

Test Türü	ASYNC_ON TPS	ASYNC_OFF TPS	Fark (%)
Read-Write	2198.149580	2130.061406	3.20
Read-Only	3901.286391	3979.468599	-1.96
Write	2089.783395	2006.823348	4.13

pg_upgrade artık istatistiklerini sıfırlamıyor

- **PG17:**
  - set default_statistics_target TO 10000;
  - analyze;
- **PG18:**
  - --no-statistics

B-tree - Skip Scan Becerisi

```
                                  Table "public.transactions"
   Column    |     Type      | Collation | Nullable |                  Default
-------------+---------------+-----------+----------+-----------------------------------------
 id          | bigint        |           | not null | nextval('transactions_id_seq'::regclass)
 region_id   | integer       |           |          |
 store_id    | integer       |           |          |
 customer_id | integer       |           |          |
 order_date  | date          |           |          |
 amount      | numeric(10,2) |           |          |
Indexes:
    "transactions_pkey" PRIMARY KEY, btree (id)
    "transactions_store_id_customer_id_order_date_idx" btree (store_id, customer_id, order_date)

Count: 5000000
```

B-tree - Skip Scan Becerisi

PG17:

```
 Gather  (cost=1000.00..69214.01 rows=1656 width=30) (actual time=2.509..530.885 rows=854 loops=1)
   Output: id, region_id, store_id, customer_id, order_date, amount
   Buffers: shared read=36800
   ->  Parallel Seq Scan on public.transactions  (cost=0.00..68048.41 rows=690 width=30) (actual
time=2.048..530.170 rows=854 loops=1)
         Output: id, region_id, store_id, customer_id, order_date, amount
         Filter: ((transactions.customer_id > 4999999) AND (transactions.order_date = '2020-01-01'::date))
         Rows Removed by Filter: 4999146
         Buffers: shared read=36800
 Query Identifier: 8124620765875735481
 Planning:
   Buffers: shared hit=86 read=21
 Planning Time: 0.918 ms
 Execution Time: 531.100 ms
```

B-tree - Skip Scan Becerisi

PG18:

```
Index Scan using transactions_store_id_customer_id_order_date_idx on public.transactions
(cost=0.43..35848.10 rows=1651 width=30) (actual time=0.244..205.854 rows=854.00 loops=1)
   Output: id, region_id, store_id, customer_id, order_date, amount
   Index Cond: ((transactions.customer_id > 4999999) AND (transactions.order_date = '2020-01-01'::date))
   Index Searches: 1001
   Buffers: shared hit=1955 read=12453
 Query Identifier: 4256613925816824965
 Planning:
   Buffers: shared hit=83 read=24
 Planning Time: 1.030 ms
 Execution Time: 206.078 ms
(10 rows)
```

B-tree - Skip Scan Becerisi

Öncül sütunda eşitlik koşulu olmasa bile, sonraki sütun koşullarını kullanarak indeksin daha küçük bir kısmını okumayı mümkün kılıyor. Yani, indeksin bir kısmını atlayarak tarama yapılabiliyor.

UUIDv7

- UUID v4
  - Fully random
- UUID v7
  - Time-ordered. 48-bit timestamp (milliseconds) + random bits.

UUIDv7

Functions

- uuid_extract_timestamp(uuid)
  - uuid içinden zaman bilgisini çıkarır.
- uuid_extract_version(uuid)
  - UUID sürümünü döner.
- uuidv7 ([shift interval]) -- 1 saat sonrası için uuid oluşturur.

“Virtual” Generated Columns

PG17:

```
CREATE TABLE products (
    id SERIAL PRIMARY KEY,
    price NUMERIC(10,2),
    tax_rate NUMERIC(3,2),
    total_price NUMERIC(10,2)
        GENERATED ALWAYS AS (price * (1 + tax_rate)) STORED
);
```

“Virtual” Generated Columns

PG18:

```
CREATE TABLE products (
    id SERIAL PRIMARY KEY,
    price NUMERIC(10,2),
    tax_rate NUMERIC(3,2),
    total_price_stored NUMERIC(10,2)
        GENERATED ALWAYS AS (price * (1 + tax_rate)) STORED,
    total_price NUMERIC(10,2)
        GENERATED ALWAYS AS (price * (1 + tax_rate)) -- By default
);
```

OAuth Authentication

pg_hba.conf

```
# TYPE   DATABASE   USER       ADDRESS          METHOD   OPTIONS
host     my_db      my_user    10.10.0.0/16     oauth    scope=".." issuer=https://../auth
validator=... map="my_map"
```

ident.conf

```
# MAPNAME        SYSTEM-USERNAME          PG-USERNAME
my_map     abcdef_my_user_id        tester
```

postgresql.conf

```
oauth_validator_libraries = 'my_validator'
```

OAuth Authentication

```
psql
"postgres://10.10.0.12/dbname?oauth_issuer=https://…/auth&oauth_client_id=my-app-client-id"
```

“OLD” and “NEW” support for RETURNING clauses

Audit Logging - Race Condition - Change Data Capture

- UPDATE
- DELETE
- INSERT
- MERGE

OLD: Bir DML operasyonundan önceki satır durumunu ifade eder.

NEW: Bir DML operasyonundan sonraki satır durumunu ifade eder.

“OLD” and “NEW” support for RETURNING clauses

PostgreSQL 18 Öncesi (Self-Join Çözümü)

```
UPDATE
  products p_new
SET
  quantity = 300
FROM products p_old
WHERE
  p_new.id = p_old.id
  AND p_new.id = 5
 RETURNING
  p_new.product_name,
  p_old.quantity,
  p_new.quantity;
```

PostgreSQL 18 (OLD/NEW Sözdizimi)

```
UPDATE
  products
SET
  quantity = 300
WHERE
  id = 5
RETURNING product_name,
  old.quantity,
  new.quantity;
```

“OLD” and “NEW” support for RETURNING clauses

PostgreSQL 18 Öncesi

```
MERGE INTO inventory AS target
USING stage_updates AS source
ON target.sku = source.sku
WHEN MATCHED THEN
    UPDATE SET quantity = source.quantity
WHEN NOT MATCHED THEN
    INSERT (sku, quantity)
    VALUES (source.sku, source.quantity);
```

PostgreSQL 18

```
MERGE INTO inventory AS target
USING stage_updates AS source
ON (target.sku = source.sku)
WHEN MATCHED THEN
    UPDATE SET quantity = source.quantity
WHEN NOT MATCHED THEN
    INSERT (sku, quantity)
    VALUES (source.sku, source.quantity)
RETURNING
    merge_action() AS action,
    source.sku,
    OLD.quantity AS old_quantity,
    NEW.quantity AS new_quantity;
```

Temporal Constraints (WITHOUT OVERLAPS)

Allow the specification of non-overlapping PRIMARY KEY, UNIQUE, and foreign key constraints

```
UNIQUE [ NULLS [ NOT ] DISTINCT ] (column constraint)

UNIQUE [ NULLS [ NOT ] DISTINCT ] ( column_name [, ... ] [, column_name WITHOUT OVERLAPS ] ) [
INCLUDE ( column_name [, ...]) ]
```

- PG17:
  - btree_gist
  - EXCLUDE USING GIST (group_id WITH =, valid_at WITH &&)
- PG18:
  - UNIQUE (group_id, valid_at WITHOUT OVERLAPS) b

Temporal Constraints (WITHOUT OVERLAPS)

Nerelerde Kullanılır?

- sözleşme/abonelik dönemleri
- koltuk rezervasyonu
- fiyat geçerlilik aralıkları
- kiralama slotları

Temporal Constraints (WITHOUT OVERLAPS)

```
postgres=# CREATE EXTENSION IF NOT EXISTS btree_gist;
CREATE EXTENSION
postgres=# CREATE TABLE reservations (
      room_id INTEGER,
      reservation_dates DATERANGE,
      UNIQUE (room_id, reservation_dates WITHOUT OVERLAPS));

postgres=# INSERT INTO reservations VALUES
(1, '[2024-01-01, 2024-01-05)'),  -- ✅ Başarılı
(1, '[2024-01-05, 2024-01-10)'),  -- ✅ Başarılı
(1, '[2024-01-03, 2024-01-07)');  -- ❌ Hata (çakışıyor)
ERROR:  conflicting key value violates exclusion constraint
"reservations_room_id_reservation_dates_key"
DETAIL:  Key (room_id, reservation_dates)=(1, [2024-01-03,2024-01-07)) conflicts with existing
key (room_id, reservation_dates)=(1, [2024-01-01,2024-01-05)).
```

Temporal Constraints (WITHOUT OVERLAPS)

Normal UNIQUE: NULL'lar birbirine eşit sayılmaz, çok sayıda NULL kabul edilir

```
postgres=# CREATE TABLE uniq_nulls_demo (
      k int,
      note text,
      UNIQUE (k)
);
CREATE TABLE
postgres=# INSERT INTO uniq_nulls_demo VALUES
(NULL, 'a'), (NULL, 'b');
INSERT 0 2
```

Temporal Constraints (WITHOUT OVERLAPS)

UNIQUE NULLS NOT DISTINCT

```
postgres=# CREATE TABLE uniq_nulls_strict (
      k int,
      note text,
      UNIQUE NULLS NOT DISTINCT (k)
);
CREATE TABLE
postgres=# INSERT INTO uniq_nulls_strict VALUES (NULL, 'a');
INSERT 0 1
postgres=# INSERT INTO uniq_nulls_strict VALUES (NULL, 'b');
ERROR:  duplicate key value violates unique constraint "uniq_nulls_strict_k_key"
DETAIL:  Key (k)=(null) already exists.
```

Initdb - data checksums

- **PG17:**
  - `initdb … --data-checksums`
- **PG18:**
  - `initdb … --no-data-checksums`

VACUUM & ANALYZE

- **PG17:**
  - `VACUUM table_name;`
  - `ANALYZE table_name;`
- **PG18:**
  - `VACUUM [ONLY] table_name [*];`
  - `ANALYZE [ONLY] table_name [*];`

Allow LIKE With Nondeterministic Collations

```
CREATE COLLATION tr_ci (provider = icu,locale = 'tr-u-ks-level2',
deterministic = false);


CREATE TABLE test_tr_ci (id serial PRIMARY KEY,name text COLLATE tr_ci);


INSERT INTO test_tr_ci (name)

VALUES ('İstanbul'),('istanbul'),('Izmir'),('ızmir'),('Ankara'),
('ankara'),('Çanakkale'), ('çanakkale');
```

Allow LIKE With Nondeterministic Collations

PG17:

```
postgres=# SELECT * FROM test_tr_ci WHERE name LIKE 'ç%';
ERROR:  nondeterministic collations are not supported for LIKE
```

PG18:

```
postgres=# SELECT * FROM test_tr_ci WHERE name LIKE 'ç%';
 id |   name
----+-----------
  7 | Çanakkale
  8 | çanakkale
(2 rows)
```

COPY FROM ... REJECT_LIMIT

```
postgres=#  copy t from '/home/postgres/load.txt' with ( delimiter ',', on_error ignore,
reject_limit 1);

ERROR:  skipped more than REJECT_LIMIT (1) rows due to data type incompatibility

CONTEXT:  COPY t, line 8, column b: "eight"

postgres=#  copy t from '/home/postgres/load.txt' with ( delimiter ',', on_error ignore,
reject_limit 2);

ERROR:  skipped more than REJECT_LIMIT (2) rows due to data type incompatibility

CONTEXT:  COPY t, line 10, column a: ""

postgres=#  copy t from '/home/postgres/load.txt' with ( delimiter ',', on_error ignore,
reject_limit 3);

COPY 7
```

COPY FROM ... REJECT_LIMIT

```
postgres=# select * from t;
 a | b |   c
---+---+-------
 1 | 1 | 'aaa'
 2 | 2 | 'bbb'
 3 | 3 | 'ccc'
(4 is incorrect)
 5 | 5 | 'eee'
 6 | 6 | 'fff'
 7 | 7 | 'ggg'
(8 is incorrect)
 9 | 9 | 'iii'
(7 rows)
```

Bonus

- FOREIGN KEY - [ENFORCED | NOT ENFORCED]
- Deprecate MD5 password
- PG13 deprecated

Teşekkürler

Varsa soruları alalım?

İletişim

- @sahapasci
- sahap.asci@cooksoft.com.tr

Bağlantılar

- Chapter 43.13. Porting from Oracle PL/SQL
- ora2Pg
- orafce
- oracle_fdw
- sqlserver2pgsql
- mysql_fdw
- pgloader
- tds_fdw
- Debezium
- JDBC Sink Connector