ΣΥΝΟΔΕΥΤΙΚΟΣ ΤΟΜΟΣ ΤΟΥ «POSTGRESQL ΣΕ PRODUCTION»

PostgreSQL

Λύσεις και εξηγήσεις

Κάθε λύση εκτελέστηκε σε πραγματικούς servers PostgreSQL 18.6.

142 **λύσεις σε 26 κεφάλαια**

Η ΕΝΤΟΛΗ, Η ΕΞΟΔΟΣ ΚΑΙ ΤΟ ΓΙΑΤΙ

ΠΩΣ ΝΑ ΧΡΗΣΙΜΟΠΟΙΗΣΕΙΣ ΑΥΤΟΝ ΤΟΝ ΤΟΜΟ

Πρώτα η δική σου προσπάθεια. Μετά η σύγκριση.

Άνοιξε μια λύση αφού το query σου δώσει το αναμενόμενο αποτέλεσμα ή αφού κολλήσεις για αρκετή ώρα. Μια διαφορετική λύση που δίνει τις ίδιες γραμμές είναι συνήθως εξίσου σωστή. Η εξήγηση κάτω από κάθε λύση λέει τι πρέπει να ισχύει για να είναι σωστή και, όπου υπάρχει, ποια κοινή παγίδα αποφεύγει.

Οι λύσεις σε SQL που αλλάζουν δεδομένα εκτελέστηκαν μέσα σε transaction που έγινε rollback, εκτός αν η λύση χρειάζεται commit, όπως ένα `VACUUM` ή ένα `LISTEN`. Οι λύσεις με εντολές shell εκτελέστηκαν με τη σειρά, στους servers του εργαστηρίου όπως τους άφησε η θεωρία και οι προηγούμενες ασκήσεις του κεφαλαίου. Αν ένα πείραμά σου αλλάξει την κατάσταση, το `pglab rm` και οι εντολές στην αρχή του κεφαλαίου σε φέρνουν πίσω.

Ο ΧΑΡΤΗΣ ΤΟΥ ΒΙΒΛΙΟΥ

Περιεχόμενα

ΚΕΦΑΛΑΙΟ 1

Domains, enums και composite types

Άσκηση 1.1 · Κινητό τηλέφωνο

Φτιάξε domain `mobile_gr` πάνω σε `text` που δέχεται μόνο δέκα ψηφία που ξεκινούν από `69`. Δοκίμασε τη μετατροπή του `'6944123456'` και του `'2105550101'` στο domain σε δύο χωριστές εντολές.

ΛΥΣΗ

```
CREATE DOMAIN mobile_gr AS text CHECK (VALUE ~ '^69[0-9]{8}$');

SELECT '6944123456'::mobile_gr AS ok;
SELECT '2105550101'::mobile_gr AS not_ok;
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE DOMAIN

     ok
------------
 6944123456
(1 row)

ERROR:  value for domain mobile_gr violates check constraint "mobile_gr_check"
```

Η πρώτη μετατροπή περνά. Η δεύτερη αποτυγχάνει, γιατί το τηλέφωνο είναι σταθερό της Αθήνας. Όλα τα κινητά των πελατών του βιβλίου ταιριάζουν στον κανόνα, οπότε το domain θα μπορούσε να μπει στη στήλη `customers.phone`.

Άσκηση 1.2 · Κανόνας που δεν τηρούν όλοι

Φτιάξε domain `phone_digits` πάνω σε `text` με `CHECK (VALUE ~ '^[0-9]+$')`. Βρες πρώτα ποιοι προμηθευτές έχουν τηλέφωνο που δεν τηρεί τον κανόνα και μετά δοκίμασε να αλλάξεις τον τύπο της στήλης `suppliers.phone` σε `phone_digits`.

ΛΥΣΗ

```
CREATE DOMAIN phone_digits AS text CHECK (VALUE ~ '^[0-9]+$');

SELECT id, name, phone FROM suppliers WHERE phone !~ '^[0-9]+$';

ALTER TABLE suppliers ALTER COLUMN phone TYPE phone_digits;
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE DOMAIN

 id |     name      |    phone
----+---------------+--------------
  5 | Nordic Office | +46855500404
(1 row)

ERROR:  value for domain phone_digits violates check constraint "phone_digits_check"
```

Το query βρίσκει έναν προμηθευτή με διεθνή μορφή και `+` μπροστά. Η αλλαγή τύπου ελέγχει κάθε υπάρχουσα τιμή και αποτυγχάνει ακριβώς σε αυτό το τηλέφωνο. Ή αλλάζεις τον κανόνα ώστε να δέχεται το `+` ή διορθώνεις πρώτα τα δεδομένα.

Άσκηση 1.3 · Σειρά ταξινόμησης από enum

Φτιάξε enum `shipping_speed` με τιμές `economy`, `standard` και `express`, με αυτή τη σειρά. Σε ένα `VALUES` με τις τρεις τιμές ως text, ανακατεμένες, ταξινόμησέ τες από την πιο γρήγορη στην πιο αργή.

ΛΥΣΗ

```
CREATE TYPE shipping_speed AS ENUM ('economy', 'standard', 'express');

SELECT speed
FROM (VALUES ('standard'), ('express'), ('economy')) AS v(speed)
ORDER BY speed::shipping_speed DESC;
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE TYPE

  speed
-----------
 express
 standard
 economy
(3 rows)
```

Χωρίς το cast η ταξινόμηση θα ήταν αλφαβητική. Το cast σε enum δίνει τη σειρά της δήλωσης, οπότε το `DESC` βάζει πρώτο το `express`.

Άσκηση 1.4 · Νέα προτεραιότητα στην αρχή

Πρόσθεσε στο enum `ticket_priority` του κεφαλαίου την τιμή `trivial` πριν από το `low` και δείξε τη νέα λίστα τιμών με τη σειρά τους.

ΛΥΣΗ

```
ALTER TYPE ticket_priority ADD VALUE 'trivial' BEFORE 'low';

SELECT unnest(enum_range(NULL::ticket_priority)) AS priority;
```

ΑΠΟΤΕΛΕΣΜΑ

```
ALTER TYPE

 priority
----------
 trivial
 low
 normal
 elevated
 high
 urgent
(6 rows)
```

Το `BEFORE` και το `AFTER` ορίζουν τη θέση. Το `unnest` απλώνει τον πίνακα που επιστρέφει το `enum_range` σε γραμμές. Αν τρέξεις τις δύο εντολές μέσα στο ίδιο `BEGIN`, το `SELECT` αποτυγχάνει με `unsafe use of new value`. Μια νέα τιμή enum χρησιμοποιείται μόνο αφού γίνει commit το `ALTER TYPE`.

Άσκηση 1.5 · Ολόκληρη η παραγγελία σε JSON

Επέστρεψε την παραγγελία 5 ως ένα JSON αντικείμενο που περιέχει όλες τις στήλες της γραμμής του `orders`, με τη χρήση του row type του πίνακα.

ΛΥΣΗ

```
SELECT jsonb_pretty(to_jsonb(o)) AS order_json
FROM orders AS o
WHERE o.id = 5;
```

ΑΠΟΤΕΛΕΣΜΑ

```
                  order_json
-----------------------------------------------
 {                                             +
     "id": 5,                                  +
     "status": "delivered",                    +
     "created_at": "2025-02-06T22:45:00+02:00",+
     "shipped_at": "2025-02-11T00:45:00+02:00",+
     "coupon_code": null,                      +
     "customer_id": 5,                         +
     "shipping_city_id": 4                     +
 }
(1 row)
```

Το alias `o` μόνο του είναι ολόκληρη η γραμμή. Αν αύριο προστεθεί στήλη στον πίνακα, θα εμφανιστεί και στο JSON χωρίς αλλαγή στο query.

Άσκηση 1.6 · Σύνοψη παραγγελίας με composite type

Φτιάξε composite type `order_summary` με πεδία `items integer` και `total numeric`. Φτιάξε function `summarize_order(p_order integer)` που επιστρέφει αυτόν τον τύπο με το πλήθος των γραμμών και το σύνολο `quantity * unit_price` της παραγγελίας. Κάλεσέ τη στο `FROM` για την παραγγελία 5.

ΛΥΣΗ

```
CREATE TYPE order_summary AS (items integer, total numeric);

CREATE FUNCTION summarize_order(p_order integer)
RETURNS order_summary
LANGUAGE sql
STABLE
RETURN (SELECT ROW(count(*), sum(quantity * unit_price))::order_summary
        FROM order_items WHERE order_id = p_order);

SELECT s.items, s.total FROM summarize_order(5) AS s;
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE TYPE

CREATE FUNCTION

 items | total
-------+-------
     1 | 34.90
(1 row)
```

Το `count(*)` επιστρέφει `bigint` και το `ROW(...)::order_summary` το μετατρέπει σε `integer`. Στο `FROM` η function εκτελείται μία φορά και τα πεδία της γίνονται στήλες.

ΚΕΦΑΛΑΙΟ 2

Prepared statements και generic plans

Άσκηση 2.1 · Προϊόντα κατηγορίας

Φτιάξε prepared statement `category_products` με μία παράμετρο `integer` που επιστρέφει όνομα και τιμή των ενεργών προϊόντων μιας κατηγορίας, με φθηνότερο πρώτο. Εκτέλεσέ το για την κατηγορία 9.

ΛΥΣΗ

```
PREPARE category_products(integer) AS
  SELECT name, price FROM products
  WHERE category_id = $1 AND active
  ORDER BY price;

EXECUTE category_products(9);
```

ΑΠΟΤΕΛΕΣΜΑ

```
PREPARE

         name          | price
-----------------------+--------
 Ασύρματο ποντίκι      |  24.90
 Webcam HD             |  59.00
 Μηχανικό πληκτρολόγιο |  89.00
 USB-C docking station | 139.00
 Οθόνη 27" 4K          | 329.00
(5 rows)
```

Το `PREPARE` δεν αναιρείται με `ROLLBACK`. Το statement μένει στη σύνδεση ακόμη κι αν το transaction όπου φτιάχτηκε ακυρωθεί. Μόνο το `DEALLOCATE` ή το κλείσιμο της σύνδεσης το σβήνουν.

Άσκηση 2.2 · Πάντα custom για τα σπάνια

Φτιάξε prepared statement `status_count(text)` με `SELECT count(*) FROM lab.orders WHERE status = $1` και εκτέλεσέ το έξι φορές για το `pending`. Δείξε πόσα custom και πόσα generic plans έγιναν.

ΛΥΣΗ

```
PREPARE status_count(text) AS
  SELECT count(*) FROM lab.orders WHERE status = $1;

EXECUTE status_count('pending');
EXECUTE status_count('pending');
EXECUTE status_count('pending');
EXECUTE status_count('pending');
EXECUTE status_count('pending');
EXECUTE status_count('pending');

SELECT generic_plans, custom_plans
FROM pg_prepared_statements WHERE name = 'status_count';
```

ΑΠΟΤΕΛΕΣΜΑ

```
PREPARE

 count
-------
  1349
(1 row)

 count
-------
  1349
(1 row)

 count
-------
  1349
(1 row)

 count
-------
  1349
(1 row)

 count
-------
  1349
(1 row)

 count
-------
  1349
(1 row)

 generic_plans | custom_plans
---------------+--------------
             0 |            6
(1 row)
```

Και οι έξι εκτελέσεις πήραν custom plan. Τα custom plans για το `pending` είναι φθηνά index scans, ενώ το generic plan υποθέτει ένα έκτο του πίνακα και κοστίζει πολύ περισσότερο. Η σύγκριση της έκτης

εκτέλεσης το απέρριψε. Η σειρά των πρώτων τιμών καθορίζει τι θα γίνει, κάτι που εξηγεί γιατί το ίδιο query συμπεριφέρεται διαφορετικά σε διαφορετικές συνδέσεις.

Άσκηση 2.3 · Generic plan με δύο παραμέτρους

Δείξε χωρίς εκτέλεση και χωρίς κόστη το generic plan του query που μετρά τις παραγγελίες του `lab.orders` με `total` ανάμεσα σε `$1` και `$2`.

ΛΥΣΗ

```
EXPLAIN (GENERIC_PLAN, COSTS OFF)
SELECT count(*) FROM lab.orders WHERE total BETWEEN $1 AND $2;
```

ΑΠΟΤΕΛΕΣΜΑ

```
Finalize Aggregate
  ->  Gather
        Workers Planned: 2
        ->  Partial Aggregate
              ->  Parallel Seq Scan on orders
                    Filter: ((total >= $1) AND (total <= $2))
```

Χωρίς index στο `total` ο planner έχει μία επιλογή, parallel sequential scan. Οι παράμετροι εμφανίζονται στο `Filter` ως `$1` και `$2`.

Άσκηση 2.4 · Επιβολή generic plan

Με `plan_cache_mode = force_generic_plan` δείξε με `EXPLAIN (COSTS OFF)` το plan του `by_status` για την τιμή `refunded`, χωρίς να αφήσεις τη ρύθμιση αλλαγμένη.

ΛΥΣΗ

```
SET plan_cache_mode = force_generic_plan;
EXPLAIN (COSTS OFF) EXECUTE by_status('refunded');
RESET plan_cache_mode;
```

ΑΠΟΤΕΛΕΣΜΑ

```
SET

Finalize Aggregate
  ->  Gather
        Workers Planned: 2
        ->  Partial Aggregate
              ->  Parallel Bitmap Heap Scan on orders
                    Recheck Cond: (status = $1)
                    ->  Bitmap Index Scan on orders_status_idx
                          Index Cond: (status = $1)

RESET
```

Το plan έχει `$1` στη συνθήκη, άρα είναι generic. Είναι το ίδιο plan που χρησιμοποιήθηκε και για το `pending` και για το `delivered`, αφού δεν εξαρτάται από την τιμή.

Άσκηση 2.5 · Στήλη που προστέθηκε

Φτιάξε prepared statement `supplier_row(integer)` με `SELECT * FROM suppliers WHERE id = $1`, εκτέλεσέ το για τον προμηθευτή 1, πρόσθεσε στήλη `rating smallint` στον `suppliers` και εκτέλεσέ το ξανά.

ΛΥΣΗ

```
PREPARE supplier_row(integer) AS SELECT * FROM suppliers WHERE id = $1;
EXECUTE supplier_row(1);

ALTER TABLE suppliers ADD COLUMN rating smallint;
EXECUTE supplier_row(1);
```

ΑΠΟΤΕΛΕΣΜΑ

```
PREPARE

 id |         name          | city_id |              email               |   phone
----+-----------------------+---------+----------------------------------+------------
  1 | Βιβλιοδιανομή Αττικής |       1 | orders@vivliodianomi.example.gr | 2105550101
(1 row)

ALTER TABLE

ERROR:  cached plan must not change result type
```

Η δεύτερη εκτέλεση αποτυγχάνει με `cached plan must not change result type`. Ένα statement με ρητή λίστα στηλών, όπως `SELECT id, name, email`, δεν θα επηρεαζόταν από τη νέα στήλη.

Άσκηση 2.6 · Καθαρή σύνδεση

Σβήσε όλα τα prepared statements της σύνδεσης και δείξε ότι το `pg_prepared_statements` είναι άδειο.

ΛΥΣΗ

```
DEALLOCATE ALL;
SELECT count(*) AS prepared FROM pg_prepared_statements;
```

ΑΠΟΤΕΛΕΣΜΑ

```
DEALLOCATE ALL

 prepared
----------
        0
(1 row)
```

Κάτι αντίστοιχο κάνει το PgBouncer σε session mode. Όταν ένας client αφήνει μια σύνδεση, στέλνει `DISCARD ALL`, που περιλαμβάνει το `DEALLOCATE ALL`, ώστε ο επόμενος client να βρει καθαρή σύνδεση.

ΚΕΦΑΛΑΙΟ 3

Dynamic SQL με EXECUTE και format

Άσκηση 3.1 · Quoting για κάθε περίπτωση

Με ένα `format` φτιάξε την εντολή που αλλάζει σε `Nikos` το όνομα των γραμμών του πίνακα `Πελάτες 2026` με email `nikos'o@example.gr`, στη στήλη `first name`. Μην την εκτελέσεις, επέστρεψε μόνο το κείμενο.

ΛΥΣΗ

```
SELECT format('UPDATE %I SET %I = %L WHERE email = %L',
              'Πελάτες 2026', 'first name', 'Nikos', 'nikos''o@example.gr') AS sql;
```

ΑΠΟΤΕΛΕΣΜΑ

```
                                        sql
-------------------------------------------------------------------------------
 UPDATE "Πελάτες 2026" SET "first name" = 'Nikos' WHERE email = 'nikos''o@example.gr'
(1 row)
```

Και τα δύο ονόματα πήραν διπλά εισαγωγικά, γιατί έχουν κενό. Οι τιμές πήραν μονά και το εισαγωγικό μέσα στο email διπλασιάστηκε.

Άσκηση 3.2 · Μέγιστη τιμή οποιασδήποτε στήλης

Φτιάξε function `max_of(p_table regclass, p_column text)` που επιστρέφει ως `text` τη μέγιστη τιμή της στήλης. Κάλεσέ τη για το `price` του `products` και για το `created_at` του `lab.orders`.

ΛΥΣΗ

```
CREATE FUNCTION max_of(p_table regclass, p_column text)
RETURNS text
LANGUAGE plpgsql
AS $$
DECLARE
  result text;
BEGIN
  EXECUTE format('SELECT max(%I)::text FROM %s', p_column, p_table) INTO result;
  RETURN result;
END;
$$;

SELECT max_of('products', 'price') AS max_price,
       max_of('lab.orders', 'created_at') AS last_order;
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE FUNCTION

 max_price |          last_order
-----------+-------------------------------
 2199.00   | 2026-07-01 02:44:55.05438+03
(1 row)
```

Ο πίνακας μπαίνει με `%s`, αφού ως `regclass` είναι ήδη σωστά γραμμένος. Η στήλη μπαίνει με `%I`. Το cast σε `text` γίνεται μέσα στο query, ώστε η function να δουλεύει για στήλες κάθε τύπου.

Άσκηση 3.3 · Πελάτες ανά στήλη με ασφάλεια

Φτιάξε function `customers_matching(p_column text, p_value text)` που επιστρέφει `id` και `email` των πελατών όπου η στήλη ισούται με την τιμή. Επίτρεψε μόνο τις στήλες `last_name` και `phone`. Κάλεσέ τη για `last_name = 'Παπαδόπουλος'`.

ΛΥΣΗ

```
CREATE FUNCTION customers_matching(p_column text, p_value text)
RETURNS TABLE (id integer, email text)
LANGUAGE plpgsql
AS $$
BEGIN
  IF p_column NOT IN ('last_name', 'phone') THEN
    RAISE EXCEPTION 'μη επιτρεπτή στήλη %', p_column;
  END IF;
  RETURN QUERY EXECUTE
    format('SELECT id, email FROM customers WHERE %I = $1 ORDER BY id', p_column)
    USING p_value;
END;
$$;

SELECT * FROM customers_matching('last_name', 'Παπαδόπουλος');
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE FUNCTION

 id |       email
----+-------------------
  2 | gpapad@example.gr
(1 row)
```

Η στήλη ελέγχεται με λίστα και μπαίνει με `%I`. Η τιμή περνά με `USING` και δεν γίνεται ποτέ μέρος του κειμένου.

Άσκηση 3.4 · Πίνακες ανά τρίμηνο

Με ένα `DO` block φτιάξε τέσσερις άδειους πίνακες `payments_2025_q1` έως `payments_2025_q4` με τις ίδιες στήλες με τον `payments`. Μετά μέτρησε πόσοι πίνακες υπάρχουν με όνομα που αρχίζει από `payments_2025`.

ΛΥΣΗ

```
DO $$
BEGIN
  FOR q IN 1..4 LOOP
    EXECUTE format('CREATE TABLE %I (LIKE payments)', 'payments_2025_q' || q);
  END LOOP;
END;
$$;

SELECT count(*) AS tables
FROM pg_tables
WHERE tablename LIKE 'payments\_2025%';
```

ΑΠΟΤΕΛΕΣΜΑ

```
DO

 tables
--------
      4
(1 row)
```

Η μεταβλητή ενός `FOR` σε διάστημα ακεραίων δηλώνεται αυτόματα, οπότε δεν χρειάστηκε `DECLARE`. Το `LIKE payments` αντιγράφει στήλες και τύπους. Το `\_` στο `LIKE` της αναζήτησης ζητά κυριολεκτική κάτω παύλα, αφού αλλιώς η `_` ταιριάζει με οποιονδήποτε χαρακτήρα.

Άσκηση 3.5 · Ένωση πολλών πινάκων

Φτιάξε με `string_agg` και `format` ένα query κειμένου που ενώνει με `UNION ALL` ένα `SELECT` ανά πίνακα του schema `lab`, όπου κάθε `SELECT` επιστρέφει το όνομα του πίνακα και το `count(*)` του. Επέστρεψε μόνο το κείμενο, ταξινομημένο κατά όνομα πίνακα.

ΛΥΣΗ

```
SELECT string_agg(
         format('SELECT %L AS table_name, count(*) FROM lab.%I', tablename, tablename),
         E'\nUNION ALL\n' ORDER BY tablename) AS sql
FROM pg_tables
WHERE schemaname = 'lab';
```

ΑΠΟΤΕΛΕΣΜΑ

```
                              sql
-----------------------------------------------------------------
 SELECT 'customers' AS table_name, count(*) FROM lab.customers+
 UNION ALL                                                    +
 SELECT 'events' AS table_name, count(*) FROM lab.events      +
 UNION ALL                                                    +
 SELECT 'orders' AS table_name, count(*) FROM lab.orders      +
 UNION ALL                                                    +
 SELECT 'products' AS table_name, count(*) FROM lab.products
(1 row)
```

Το όνομα του πίνακα εμφανίζεται δύο φορές με δύο ρόλους. Ως τιμή στη λίστα του `SELECT` μπαίνει με `%L` και ως πίνακας στο `FROM` με `%I`. Το κείμενο αυτό μπορεί να εκτελεστεί με `EXECUTE` μέσα σε function ή απευθείας με `\gexec`.

Άσκηση 3.6 · Injection στην πράξη

Χρησιμοποίησε την `count_status_unsafe` του κεφαλαίου για να μετρήσεις τις παραγγελίες που δεν έχουν κατάσταση `delivered`, περνώντας μόνο ένα κατάλληλο string ως όρισμα.

ΛΥΣΗ

```
SELECT count_status_unsafe($$x' OR status <> 'delivered$$) AS not_delivered;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 not_delivered
---------------
            37
(1 row)
```

Το string κλείνει την τιμή με το πρώτο εισαγωγικό και αφήνει το τελευταίο εισαγωγικό της function να κλείσει το `'delivered'`. Το εκτελεσμένο query έγινε `status = 'x' OR status <> 'delivered'`. Όποιος ελέγχει την τιμή ελέγχει το query.

ΚΕΦΑΛΑΙΟ 4

LISTEN και NOTIFY

Άσκηση 4.1 · Ειδοποίηση χωρίς payload

Στη συνεδρία Α άκου στο κανάλι `cache_reset` και στείλε σε αυτό μια ειδοποίηση χωρίς payload.

ΛΥΣΗ

```
LISTEN cache_reset;
NOTIFY cache_reset;
UNLISTEN cache_reset;
```

ΑΠΟΤΕΛΕΣΜΑ

```
LISTEN

NOTIFY
Asynchronous notification "cache_reset" with payload "" received from server process with PID 75732.

UNLISTEN
```

Μια σύνδεση λαμβάνει και τις δικές της ειδοποιήσεις. Εδώ η ειδοποίηση εμφανίζεται αμέσως κάτω από το `NOTIFY`. Χωρίς `BEGIN` κάθε εντολή είναι δικό της transaction, οπότε το commit έγινε με το τέλος της εντολής και η απάντηση του server έφερε μαζί και την ειδοποίηση. Το payload είναι κενό κείμενο. Πολλές εφαρμογές χρησιμοποιούν τέτοιο κανάλι για να πουν σε όλους τους servers της εφαρμογής να αδειάσουν μια cache.

Άσκηση 4.2 · Δυναμικό κανάλι

Άκου στο κανάλι `customer_7` και στείλε σε αυτό με `pg_notify` το email του πελάτη 7, φτιάχνοντας το όνομα του καναλιού μέσα στο query από το id.

ΛΥΣΗ

```
LISTEN customer_7;
SELECT pg_notify('customer_' || id, email) FROM customers WHERE id = 7;
UNLISTEN customer_7;
```

ΑΠΟΤΕΛΕΣΜΑ

```
LISTEN

 pg_notify
------------

(1 row)
Asynchronous notification "customer_7" with payload "dimitra.k@example.gr" received from
server process with PID 75732.

UNLISTEN
```

Το `NOTIFY` δεν θα δεχόταν έκφραση ως όνομα καναλιού. Το `pg_notify` δέχεται οποιαδήποτε τιμή `text`, οπότε ένα κανάλι ανά πελάτη ή ανά χρήστη φτιάχνεται με ένα query.

Άσκηση 4.3 · Ειδοποίηση για αλλαγή κατάστασης

Φτιάξε trigger στον `orders` που στέλνει στο κανάλι `order_status` payload της μορφής `id:νέα κατάσταση`, μόνο όταν αλλάζει η στήλη `status`. Άκου στο κανάλι και άλλαξε την κατάσταση της παραγγελίας 162 σε `paid`.

ΛΥΣΗ

```
CREATE FUNCTION notify_status()
RETURNS trigger
LANGUAGE plpgsql
AS $$
BEGIN
  PERFORM pg_notify('order_status', NEW.id || ':' || NEW.status);
  RETURN NULL;
END;
$$;

CREATE TRIGGER orders_status_notify
AFTER UPDATE OF status ON orders
FOR EACH ROW
WHEN (OLD.status IS DISTINCT FROM NEW.status)
EXECUTE FUNCTION notify_status();

LISTEN order_status;
UPDATE orders SET status = 'paid' WHERE id = 162;
UNLISTEN order_status;
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE FUNCTION

CREATE TRIGGER

LISTEN

UPDATE 1
Asynchronous notification "order_status" with payload "162:paid" received from server process with PID 75732.

UNLISTEN
```

Το `UPDATE OF status` ενεργοποιεί τον trigger μόνο όταν η εντολή αναφέρει τη στήλη και το `WHEN` μόνο όταν η τιμή πραγματικά άλλαξε. Ένα `UPDATE` που γράφει την ίδια κατάσταση δεν στέλνει ειδοποίηση.

Άσκηση 4.4 · Rollback χωρίς ειδοποίηση

Άκου στο κανάλι `order_status`, άλλαξε μέσα σε transaction την κατάσταση της παραγγελίας 161 σε `cancelled` και κάνε `ROLLBACK`. Δείξε μετά την κατάσταση της παραγγελίας.

ΛΥΣΗ

```
LISTEN order_status;
BEGIN;
UPDATE orders SET status = 'cancelled' WHERE id = 161;
ROLLBACK;
SELECT status FROM orders WHERE id = 161;
UNLISTEN order_status;
```

ΑΠΟΤΕΛΕΣΜΑ

```
LISTEN

BEGIN

UPDATE 1

ROLLBACK

 status
--------
 paid
(1 row)

UNLISTEN
```

Ο trigger εκτελέστηκε και κάλεσε το `pg_notify`, αλλά η ειδοποίηση περίμενε το commit που δεν ήρθε ποτέ. Σε καμία εντολή δεν εμφανίζεται ειδοποίηση, σε αντίθεση με την προηγούμενη άσκηση όπου ήρθε κάτω από το `UPDATE`.

Άσκηση 4.5 · Σε ποια κανάλια ακούω

Άκου στα κανάλια `a_jobs` και `b_jobs`, δείξε τα κανάλια της σύνδεσης ταξινομημένα και σταμάτα να ακούς σε όλα.

ΛΥΣΗ

```
LISTEN a_jobs;
LISTEN b_jobs;
SELECT channel FROM pg_listening_channels() AS channel ORDER BY channel;
UNLISTEN *;
```

ΑΠΟΤΕΛΕΣΜΑ

```
LISTEN

LISTEN

 channel
---------
 a_jobs
 b_jobs
(2 rows)

UNLISTEN
```

Το `pg_listening_channels` είναι set returning function, οπότε μπαίνει στο `FROM` σαν πίνακας. Το `UNLISTEN *` είναι αυτό που κάνει ένας pooler σε session mode πριν δώσει τη σύνδεση σε άλλον client.

ΚΕΦΑΛΑΙΟ 5

TABLESAMPLE και δείγματα

Άσκηση 5.1 · Εκτίμηση για τα checkout

Από δείγμα `BERNOULLI` του ενός τοις εκατό του `lab.events` με seed 7, εκτίμησε πόσα events έχουν `kind = 'checkout'`. Βάλε στην ίδια γραμμή και το ακριβές πλήθος.

ΛΥΣΗ

```
SELECT (SELECT count(*) * 100 FROM lab.events TABLESAMPLE BERNOULLI (1) REPEATABLE (7)
        WHERE kind = 'checkout') AS estimated,
       (SELECT count(*) FROM lab.events WHERE kind = 'checkout') AS exact;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 estimated | exact
-----------+--------
    139500 | 142762
(1 row)
```

Τα events έχουν περίπου ένα έβδομο checkout, απλωμένα ομοιόμορφα στον πίνακα. Η εκτίμηση απέχει λίγο από την ακριβή τιμή. Με `SYSTEM` θα ήταν επίσης καλή, αφού το `kind` δεν είναι μαζεμένο σε σελίδες.

Άσκηση 5.2 · Μέση αξία ανά κατάσταση

Από δείγμα `SYSTEM` του δύο τοις εκατό του `lab.orders` με seed 3, υπολόγισε τη μέση αξία παραγγελίας για τις καταστάσεις `delivered`, `cancelled` και `refunded`, με δύο δεκαδικά.

ΛΥΣΗ

```
SELECT status, round(avg(total), 2) AS avg_total, count(*) AS sampled
FROM lab.orders TABLESAMPLE SYSTEM (2) REPEATABLE (3)
WHERE status IN ('delivered', 'cancelled', 'refunded')
GROUP BY status
ORDER BY status;
```

ΑΠΟΤΕΛΕΣΜΑ

```
  status   | avg_total | sampled
-----------+-----------+---------
 cancelled |    382.14 |    2709
 delivered |    379.69 |   35695
 refunded  |    378.79 |    1170
(3 rows)
```

Αυτές οι τρεις καταστάσεις είναι απλωμένες σε όλο τον πίνακα, οπότε το `SYSTEM` δίνει λογικούς μέσους όρους με ελάχιστο κόστος. Η στήλη `sampled` δείχνει σε πόσες γραμμές βασίζεται κάθε μέσος όρος, κάτι που πάντα αξίζει να βλέπεις δίπλα σε μια εκτίμηση.

Άσκηση 5.3 · Πόσες σελίδες διαβάζει

Με `EXPLAIN (ANALYZE, BUFFERS, COSTS OFF, TIMING OFF)` δείξε πόσες σελίδες διαβάζει ένα δείγμα `SYSTEM` του μισού τοις εκατό από το `lab.events` με seed 1.

ΛΥΣΗ

```
EXPLAIN (ANALYZE, BUFFERS, COSTS OFF, TIMING OFF)
SELECT count(*) FROM lab.events TABLESAMPLE SYSTEM (0.5) REPEATABLE (1);
```

ΑΠΟΤΕΛΕΣΜΑ

```
Aggregate (actual rows=1.00 loops=1)
  Buffers: shared hit=72
  ->  Sample Scan on events (actual rows=5268.00 loops=1)
        Sampling: system ('0.5'::real) REPEATABLE ('1'::double precision)
        Buffers: shared hit=72
Planning Time: 0.026 ms
Execution Time: 0.335 ms
```

Το άθροισμα των `hit` και `read` στο `Buffers` είναι οι σελίδες του δείγματος. Είναι περίπου το μισό τοις εκατό των σελίδων του πίνακα. Το ποσοστό μπορεί να έχει δεκαδικά.

Άσκηση 5.4 · Δείγμα πελατών με τις παραγγελίες τους

Φτιάξε πίνακα `lab.customers_sample` με δείγμα `BERNOULLI` του μισού τοις εκατό των πελατών του `lab.customers` με seed 11. Μέτρησε μετά πόσους πελάτες έχει και πόσες παραγγελίες του `lab.orders` ανήκουν σε αυτούς.

ΛΥΣΗ

```
CREATE TABLE lab.customers_sample AS
SELECT * FROM lab.customers TABLESAMPLE BERNOULLI (0.5) REPEATABLE (11);

SELECT (SELECT count(*) FROM lab.customers_sample) AS customers,
       (SELECT count(*) FROM lab.orders AS o
        WHERE o.customer_id IN (SELECT id FROM lab.customers_sample)) AS orders;
```

ΑΠΟΤΕΛΕΣΜΑ

```
SELECT 1019

 customers | orders
-----------+--------
      1019 |  10533
(1 row)
```

Το δείγμα γίνεται στον γονικό πίνακα και οι παραγγελίες ακολουθούν με subquery. Έτσι κάθε παραγγελία του υποσυνόλου έχει τον πελάτη της, κάτι που δεν θα ίσχυε αν έπαιρνες χωριστό δείγμα από κάθε πίνακα.

Άσκηση 5.5 · Τρία τυχαία προϊόντα με φθηνό τρόπο

Δείξε με `EXPLAIN (COSTS OFF)` το plan ενός query που επιστρέφει τρία προϊόντα του `lab.products` με `SYSTEM_ROWS`.

ΛΥΣΗ

```
EXPLAIN (COSTS OFF)
SELECT id, name, price FROM lab.products TABLESAMPLE SYSTEM_ROWS (3);
```

ΑΠΟΤΕΛΕΣΜΑ

```
Sample Scan on products
  Sampling: system_rows ('3'::bigint)
```

Το plan είναι ένα `Sample Scan` χωρίς `Sort` και χωρίς `Limit`. Το `SYSTEM_ROWS` σταματά μόλις μαζέψει τρεις γραμμές, όσο μεγάλος κι αν είναι ο πίνακας.

Άσκηση 5.6 · Εκτίμηση από τον planner

Δείξε το `reltuples` του `lab.events` και του `lab.customers` από το `pg_class`, ως ακέραιους, με το όνομα του πίνακα.

ΛΥΣΗ

```
SELECT oid::regclass AS table_name, reltuples::bigint AS estimated_rows
FROM pg_class
WHERE oid IN ('lab.events'::regclass, 'lab.customers'::regclass)
ORDER BY 1;
```

ΑΠΟΤΕΛΕΣΜΑ

```
  table_name   | estimated_rows
---------------+----------------
 lab.customers |         200000
 lab.events    |        1000000
(2 rows)
```

Μετά το `VACUUM ANALYZE` του εργαστηρίου οι εκτιμήσεις είναι ακριβείς. Σε έναν πίνακα που αλλάζει, απέχουν από την πραγματικότητα όσο έχει αλλάξει ο πίνακας από τον τελευταίο autovacuum.

ΚΕΦΑΛΑΙΟ 6

Το εργαστήριο και το data directory

Άσκηση 6.1 · Ο φάκελος της βάσης

Βρες το OID της βάσης `sqlbook` και δείξε ότι υπάρχει υποφάκελος με αυτό το όνομα μέσα στο `base` του `primary`.

ΛΥΣΗ

```
oid=$(psql -At -c "SELECT oid FROM pg_database WHERE datname = 'sqlbook'")
echo "OID: $oid"
ls -d "$LAB/primary/base/$oid"
```

ΕΞΟΔΟΣ

```
OID: 16384
~/pglab/primary/base/16384
```

Το `-At` τυπώνει μόνο την τιμή, χωρίς επικεφαλίδες και στοίχιση, έτσι ώστε να μπει σε μεταβλητή του shell. Κάθε βάση είναι ένας φάκελος με το OID της.

Άσκηση 6.2 · Το αρχείο ενός πίνακα

Βρες με `pg_relation_filepath` σε ποιο αρχείο βρίσκεται ο πίνακας `orders` και δείξε το μέγεθός του σε bytes με `wc -c`.

ΛΥΣΗ

```
file=$(psql -At -c "SELECT pg_relation_filepath('orders')")
echo "$file"
wc -c < "$LAB/primary/$file"
```

ΕΞΟΔΟΣ

```
base/16384/16501
   16384
```

Η διαδρομή είναι σχετική με το data directory. Το μέγεθος είναι πολλαπλάσιο των 8.192 bytes, αφού ο πίνακας αποτελείται από ολόκληρες σελίδες.

Άσκηση 6.3 · Δεύτερος server

Φτιάξε με το `pglab` δεύτερο server με όνομα `second` στη θύρα 5502, ξεκίνα τον και δείξε με `pglab ls` και τους δύο. Μετά σβήσε τον.

ΛΥΣΗ

```
pglab init second 5502
pglab start second
pglab ls
pglab rm second
```

ΕΞΟΔΟΣ

```
second: νέο cluster στο $LAB/second, θύρα 5502
second: σε λειτουργία στη θύρα 5502
primary    θύρα 5501  σε λειτουργία
second     θύρα 5502  σε λειτουργία
second: σβήστηκε
```

Οι δύο servers είναι εντελώς ανεξάρτητοι, με δικό του φάκελο, θύρα, διεργασίες και μνήμη ο καθένας. Έτσι θα στήνουμε primary και standby στο μέρος IV.

Άσκηση 6.4 · Οι κανόνες πρόσβασης

Δείξε από το `pg_hba_file_rules` του `primary` τη γραμμή, τον τύπο, τη βάση και τη μέθοδο κάθε κανόνα. Μέτρησε μετά πόσοι κανόνες είναι `trust`.

ΛΥΣΗ

```
SELECT line_number, type, database, auth_method
FROM pg_hba_file_rules
ORDER BY line_number;

SELECT count(*) AS trust_rules FROM pg_hba_file_rules WHERE auth_method = 'trust';
```

ΑΠΟΤΕΛΕΣΜΑ

```
 line_number | type  |   database    | auth_method
-------------+-------+---------------+-------------
         117 | local | {all}         | trust
         119 | host  | {all}         | trust
         121 | host  | {all}         | trust
         124 | local | {replication} | trust
         125 | host  | {replication} | trust
         126 | host  | {replication} | trust
(6 rows)

 trust_rules
-------------
           6
(1 row)
```

Το `pglab init` αφήνει το αρχείο του `initdb` όπως είναι, με όλες τις γραμμές `trust`. Η στήλη `database` είναι πίνακας, γιατί μια γραμμή μπορεί να αφορά πολλές βάσεις χωρισμένες με κόμμα.

Άσκηση 6.5 · Πόσες διεργασίες ανοίγει μια σύνδεση

Άνοιξε δύο συνδέσεις στο παρασκήνιο με `psql -c "SELECT pg_sleep(2)" &` και μέτρησε από τρίτη σύνδεση πόσοι `client backend` υπάρχουν. Περίμενε μετά να τελειώσουν με `wait`.

ΛΥΣΗ

```
psql -c "SELECT pg_sleep(2)" >/dev/null &
psql -c "SELECT pg_sleep(2)" >/dev/null &
sleep 0.5
psql -At -c "SELECT count(*) FROM pg_stat_activity
             WHERE backend_type = 'client backend' AND application_name = 'psql'"
wait
```

ΕΞΟΔΟΣ

```
3
```

Τρεις client backends, δηλαδή τρεις διεργασίες, μία για κάθε `psql`. Η τρίτη είναι αυτή που μετρά. Το φίλτρο στο `application_name` αφήνει έξω άλλες συνδέσεις που ίσως έχεις ανοιχτές, όπως ένα `psql` σε άλλο terminal, αφού κάθε `psql` δηλώνει το όνομά του στον server. Το `&` τρέχει μια εντολή στο παρασκήνιο και το `wait` περιμένει όλες τις εντολές του παρασκηνίου.

Άσκηση 6.6 · Recovery μετά από πτώση

Γράψε στον `primary` έναν πίνακα `after_crash` με 1.000 γραμμές, σταμάτησέ τον με `immediate`, ξεκίνα τον και δείξε τις γραμμές του log για την αρχή και το τέλος του redo. Μέτρησε μετά τις γραμμές του πίνακα.

ΛΥΣΗ

```
psql -c "CREATE TABLE after_crash AS SELECT generate_series(1, 1000) AS n"
pg_ctl -D "$LAB/primary" -m immediate stop
pglab start primary
grep -E 'redo (starts|done)' "$LAB/primary.log" | tail -n 2
psql -At -c "SELECT count(*) FROM after_crash"
```

ΕΞΟΔΟΣ

```
SELECT 1000
waiting for server to shut down.... done
server stopped
primary: σε λειτουργία στη θύρα 5501
2026-09-28 09:46:43.751 EEST [76072] LOG:  redo starts at 0/178D118
2026-09-28 09:46:43.766 EEST [76072] LOG:  redo done at 0/1D040D8 system usage: CPU:
user: 0.00 s, system: 0.01 s, elapsed: 0.01 s
1000
```

Το `pglab stop` κάνει κανονικό σταμάτημα, γι' αυτό η λύση καλεί το `pg_ctl` με `-m immediate`. Οι γραμμές `redo starts at` και `redo done at` δείχνουν από ποιο σημείο του WAL ξεκίνησε το recovery και πού τελείωσε.

ΚΕΦΑΛΑΙΟ 7

Ρυθμίσεις και από πού έρχονται

Άσκηση 7.1 · Τι άλλαξε από τις προεπιλογές

Δείξε όλες τις ρυθμίσεις του `primary` που δεν έχουν `source` ίσο με `default`, `override` ή `client`, με το όνομα, την τιμή, τη μονάδα και την πηγή τους.

ΛΥΣΗ

```
SELECT name, setting, unit, source
FROM pg_settings
WHERE source NOT IN ('default', 'override', 'client')
ORDER BY source, name;
```

ΑΠΟΤΕΛΕΣΜΑ

```
             name             |      setting       | unit |       source
------------------------------+--------------------+------+--------------------
 autovacuum_worker_slots      | 16                 | ø    | configuration file
 DateStyle                    | ISO, MDY           | ø    | configuration file
 default_text_search_config   | pg_catalog.english | ø    | configuration file
 dynamic_shared_memory_type   | posix              | ø    | configuration file
 lc_messages                  | en_US.UTF-8        | ø    | configuration file
 lc_monetary                  | en_US.UTF-8        | ø    | configuration file
 lc_numeric                   | en_US.UTF-8        | ø    | configuration file
 lc_time                      | en_US.UTF-8        | ø    | configuration file
 listen_addresses             | localhost          | ø    | configuration file
 log_min_duration_statement   | 250                | ms   | configuration file
 log_timezone                 | Europe/Athens      | ø    | configuration file
 max_connections              | 100                | ø    | configuration file
 max_wal_size                 | 1024               | MB   | configuration file
 min_wal_size                 | 80                 | MB   | configuration file
 port                         | 5501               | ø    | configuration file
 shared_buffers               | 32768              | 8kB  | configuration file
 TimeZone                     | Europe/Athens      | ø    | configuration file
 unix_socket_directories      |                    | ø    | configuration file
 work_mem                     | 16384              | kB   | configuration file
 statement_timeout            | 30000              | ms   | database
(20 rows)
```

Αυτό το query είναι το πρώτο που τρέχεις σε έναν άγνωστο server. Σου λέει τι έχει αλλάξει κάποιος από τις προεπιλογές και σε ποιο επίπεδο. Το `override` είναι τιμές που υπολογίζει ο server στην εκκίνηση και το `client` όσες στέλνει ο client στη σύνδεση, όπως η κωδικοποίηση.

Άσκηση 7.2 · Μέγεθος σε bytes

Δείξε το `shared_buffers`, το `work_mem` και το `maintenance_work_mem` σε μορφή που διαβάζεται, με `pg_size_pretty` πάνω στο `setting` και τη μονάδα του `pg_settings`.

ΛΥΣΗ

```
SELECT name,
       pg_size_pretty(setting::bigint * CASE unit WHEN '8kB' THEN 8192
                                                  WHEN 'kB' THEN 1024 END) AS size
FROM pg_settings
WHERE name IN ('shared_buffers', 'work_mem', 'maintenance_work_mem')
ORDER BY name;
```

ΑΠΟΤΕΛΕΣΜΑ

```
         name          |  size
-----------------------+--------
 maintenance_work_mem | 64 MB
 shared_buffers       | 256 MB
 work_mem             | 16 MB
(3 rows)
```

Το `shared_buffers` μετριέται σε σελίδες των 8 KB και τα άλλα δύο σε KB. Η στήλη `unit` σου λέει πώς να μετατρέψεις το `setting`. Το `SHOW` κάνει τη μετατροπή για σένα, αλλά μόνο για μία ρύθμιση κάθε φορά.

Άσκηση 7.3 · Ρύθμιση ρόλου σε μία βάση

Φτιάξε ρόλο `batch` με login και όρισε `work_mem = '64MB'` μόνο όταν συνδέεται στη βάση `sqlbook`. Δείξε από νέα σύνδεση ως `batch` την τιμή που παίρνει.

ΛΥΣΗ

```
psql -c "CREATE ROLE batch LOGIN"
psql -c "ALTER ROLE batch IN DATABASE sqlbook SET work_mem = '64MB'"
psql -U batch -At -c "SHOW work_mem"
```

ΕΞΟΔΟΣ

```
CREATE ROLE
ALTER ROLE
64MB
```

Το `IN DATABASE` είναι το πιο ειδικό από τα επίπεδα των συνδέσεων. Ο ίδιος ρόλος σε άλλη βάση παίρνει το `work_mem` του server.

Άσκηση 7.4 · Ένα restart που περιμένει

Άλλαξε με `ALTER SYSTEM` το `max_connections` σε 150 και κάνε reload. Δείξε το `setting` και το `pending_restart` του. Κάνε restart τον `primary` και δείξε τα ξανά.

ΛΥΣΗ

```
psql -c "ALTER SYSTEM SET max_connections = 150"
psql -At -c "SELECT pg_reload_conf()"
psql -At -c "SELECT setting, pending_restart FROM pg_settings WHERE name = 'max_connections'"
pglab restart primary
psql -At -c "SELECT setting, pending_restart FROM pg_settings WHERE name = 'max_connections'"
```

ΕΞΟΔΟΣ

```
ALTER SYSTEM
t
100|t
primary: σε λειτουργία στη θύρα 5501
150|f
```

Πριν από το restart η τιμή είναι 100 και το `pending_restart` αληθές. Μετά είναι 150 και ψευδές. Το `-At` τυπώνει τις δύο στήλες χωρισμένες με `|`.

Άσκηση 7.5 · Χρόνος μόνο για ένα transaction

Μέσα σε ένα transaction όρισε `work_mem` 256 MB μόνο για αυτό το transaction και δείξε την τιμή. Μετά το `COMMIT` δείξε ξανά την τιμή.

ΛΥΣΗ

```
BEGIN;
SET LOCAL work_mem = '256MB';
SHOW work_mem;
COMMIT;
SHOW work_mem;
```

ΑΠΟΤΕΛΕΣΜΑ

```
BEGIN

SET

 work_mem
----------
 256MB
(1 row)

COMMIT

 work_mem
----------
 16MB
(1 row)
```

Το `SET LOCAL` χάνεται με το τέλος του transaction, είτε με `COMMIT` είτε με `ROLLBACK`. Ένα `SET` χωρίς `LOCAL` θα έμενε σε όλη την υπόλοιπη σύνδεση.

Άσκηση 7.6 · Ρυθμίσεις ανά context

Μέτρησε πόσες ρυθμίσεις του `pg_settings` που ξεκινούν από `autovacuum` έχει κάθε context.

ΛΥΣΗ

```
SELECT context, count(*)
FROM pg_settings
WHERE name LIKE 'autovacuum%'
GROUP BY context
ORDER BY context;
```

ΑΠΟΤΕΛΕΣΜΑ

```
  context   | count
------------+-------
 postmaster |     3
 sighup     |    13
(2 rows)
```

Οι περισσότερες ρυθμίσεις του autovacuum αλλάζουν με reload. Όσες ορίζουν πόσες διεργασίες υπάρχουν ή πόση μνήμη δεσμεύεται στην εκκίνηση θέλουν restart.

ΚΕΦΑΛΑΙΟ 8

Μνήμη

Άσκηση 8.1 · Ποιοι πίνακες είναι στη μνήμη

Δείξε για τους πίνακες του schema `lab` πόσα MB από τις σελίδες τους βρίσκονται τώρα στο `shared_buffers`, ταξινομημένα από τον μεγαλύτερο.

ΛΥΣΗ

```
SELECT c.oid::regclass AS relation,
       pg_size_pretty(count(*) * 8192) AS cached
FROM pg_buffercache AS b
JOIN pg_class AS c ON b.relfilenode = pg_relation_filenode(c.oid)
WHERE b.reldatabase = (SELECT oid FROM pg_database WHERE datname = current_database())
  AND c.relnamespace = 'lab'::regnamespace
GROUP BY c.oid
ORDER BY count(*) DESC;
```

ΑΠΟΤΕΛΕΣΜΑ

```
    relation     |   cached
-----------------+------------
 lab.customers   | 25 MB
 lab.orders      | 5264 kB
 lab.orders_pkey | 8192 bytes
(3 rows)
```

Κάθε buffer είναι 8.192 bytes, οπότε το πλήθος επί 8.192 δίνει το μέγεθος. Το `regnamespace` μετατρέπει το όνομα ενός schema στο OID του, όπως το `regclass` για τους πίνακες.

Άσκηση 8.2 · Hit ratio ανά πίνακα

Από το `pg_statio_user_tables` δείξε για τους πίνακες του `lab` τα `heap_blks_hit`, `heap_blks_read` και το ποσοστό hit με ένα δεκαδικό.

ΛΥΣΗ

```
SELECT relname, heap_blks_hit, heap_blks_read,
       round(100.0 * heap_blks_hit / nullif(heap_blks_hit + heap_blks_read, 0), 1) AS hit_pct
FROM pg_statio_user_tables
WHERE schemaname = 'lab'
ORDER BY relname;
```

ΑΠΟΤΕΛΕΣΜΑ

```
  relname  | heap_blks_hit | heap_blks_read | hit_pct
-----------+---------------+----------------+---------
 customers |       4206716 |          12998 |    99.7
 events    |       1067725 |            594 |    99.9
 orders    |       4035119 |         100703 |    97.6
 products  |         53571 |           2235 |    96.0
(4 rows)
```

Όλα τα ποσοστά είναι ψηλά λόγω της φόρτωσης, αλλά το `lab.orders` έχει πολύ περισσότερα reads από τους άλλους. Το διαβάσαμε ολόκληρο με sequential scans που χρησιμοποιούν ring buffer, οπότε σχεδόν κάθε σελίδα ήρθε από το λειτουργικό. Το `nullif` προστατεύει από διαίρεση με το μηδέν για πίνακες που δεν διαβάστηκαν ποτέ.

Άσκηση 8.3 · Ταξινόμηση που χωρά

Με `EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF, BUFFERS OFF)` δείξε το plan της ταξινόμησης των `lab.customers` κατά `last_name`, `first_name`, `email`, πρώτα με την προεπιλογή του `work_mem` και μετά με 32 MB μόνο για αυτό το transaction.

ΛΥΣΗ

```
EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF, BUFFERS OFF)
SELECT last_name, first_name, email FROM lab.customers
ORDER BY last_name, first_name, email;

BEGIN;
SET LOCAL work_mem = '32MB';
EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF, BUFFERS OFF)
SELECT last_name, first_name, email FROM lab.customers
ORDER BY last_name, first_name, email;
COMMIT;
```

ΑΠΟΤΕΛΕΣΜΑ

```
Sort (actual rows=200000.00 loops=1)
  Sort Key: last_name, first_name, email
  Sort Method: external merge  Disk: 12680kB
  ->  Seq Scan on customers (actual rows=200000.00 loops=1)
Planning Time: 0.098 ms
Execution Time: 333.913 ms

BEGIN

SET

Sort (actual rows=200000.00 loops=1)
  Sort Key: last_name, first_name, email
  Sort Method: quicksort  Memory: 20629kB
  ->  Seq Scan on customers (actual rows=200000.00 loops=1)
Planning Time: 0.023 ms
Execution Time: 354.272 ms

COMMIT
```

Με 4 MB η ταξινόμηση γίνεται `external merge` στον δίσκο. Με 32 MB γίνεται `quicksort` στη μνήμη. Το `SET LOCAL` μέσα στο transaction είναι ο τρόπος να δώσεις μνήμη σε ένα query χωρίς να την αφήσεις στη σύνδεση.

Άσκηση 8.4 · Μνήμη για τον ρόλο των αναφορών

Φτιάξε ρόλο `reports` με login και όρισε `work_mem` 128 MB για αυτόν. Δείξε από νέα σύνδεση ως `reports` την τιμή του `work_mem` και την πηγή της.

ΛΥΣΗ

```
psql -c "CREATE ROLE reports LOGIN"
psql -c "ALTER ROLE reports SET work_mem = '128MB'"
psql -U reports -c "SELECT setting, unit, source FROM pg_settings WHERE name = 'work_mem'"
```

ΕΞΟΔΟΣ

```
CREATE ROLE
ALTER ROLE
 setting | unit | source
---------+------+--------
 131072  | kB   | user
(1 row)
```

Η πηγή είναι `user`, δηλαδή η ρύθμιση του ρόλου. Οι αναφορές παίρνουν πολλή μνήμη ενώ η εφαρμογή μένει στα 4 MB και ο προϋπολογισμός του server μετράει μόνο όσες συνδέσεις του `reports` μπορεί να υπάρξουν ταυτόχρονα.

Άσκηση 8.5 · Πόσα προσωρινά αρχεία

Μηδένισε τα στατιστικά της βάσης με `pg_stat_reset()`, τρέξε ταξινόμηση όλων των `lab.orders` κατά `total` με `work_mem` 4 MB και δείξε τα `temp_files` και το μέγεθός τους από το `pg_stat_database`.

ΛΥΣΗ

```
SELECT pg_stat_reset();

SELECT id FROM lab.orders ORDER BY total OFFSET 1999999;

SELECT pg_stat_force_next_flush();

SELECT temp_files, pg_size_pretty(temp_bytes) AS temp_size
FROM pg_stat_database
WHERE datname = current_database();
```

ΑΠΟΤΕΛΕΣΜΑ

```
 pg_stat_reset
---------------

(1 row)

   id
--------
 188570
(1 row)

 pg_stat_force_next_flush
--------------------------

(1 row)

 temp_files | temp_size
------------+-----------
          2 | 60 MB
(1 row)
```

Το `OFFSET` επιστρέφει μόνο την τελευταία γραμμή, αλλά η ταξινόμηση έγινε για όλες. Τα προσωρινά αρχεία είναι τα κομμάτια της ταξινόμησης που δεν χώρεσαν στη μνήμη. Το `pg_stat_reset` μηδενίζει τους μετρητές της τρέχουσας βάσης, κάτι χρήσιμο πριν από ένα πείραμα. Κάθε διεργασία στέλνει τα στατιστικά της στην κοινή μνήμη το πολύ μία φορά το δευτερόλεπτο. Το `pg_stat_force_next_flush` τη βάζει να τα στείλει αμέσως μετά την εντολή, ώστε το τελευταίο query να τα δει.

Άσκηση 8.6 · Φόρτωση στη μνήμη

Φόρτωσε με `pg_prewarm` το primary key index του `lab.orders` και δείξε πόσες σελίδες του βρίσκονται στο `shared_buffers`.

ΛΥΣΗ

```
SELECT pg_prewarm('lab.orders_pkey') AS pages_loaded;

SELECT count(*) AS buffers
FROM pg_buffercache AS b
WHERE b.relfilenode = pg_relation_filenode('lab.orders_pkey')
  AND b.reldatabase = (SELECT oid FROM pg_database WHERE datname = current_database());
```

ΑΠΟΤΕΛΕΣΜΑ

```
 pages_loaded
--------------
         5486
(1 row)

 buffers
---------
    5486
(1 row)
```

Το `pg_prewarm` δουλεύει και για indexes. Όλες οι σελίδες που φόρτωσε είναι στη μνήμη, αφού το index των περίπου 43 MB χωρά άνετα στα 128 MB. Ένα index που χρησιμοποιείται σε κάθε αναζήτηση είναι καλός υποψήφιος για prewarm μετά από restart.

ΚΕΦΑΛΑΙΟ 9

WAL, checkpoints και durability

Άσκηση 9.1 · Το WAL ενός index

Μέτρησε πόσο WAL γράφει η δημιουργία ενός index στο `pgbench_accounts(abalance)`.

ΛΥΣΗ

```
before=$(psql -At -c "SELECT pg_current_wal_lsn()")
psql -q -c "CREATE INDEX accounts_abalance_idx ON pgbench_accounts (abalance)"
psql -At -c "SELECT pg_size_pretty(pg_wal_lsn_diff(pg_current_wal_lsn(), '$before'))"
```

ΕΞΟΔΟΣ

```
16 MB
```

Το WAL είναι περίπου όσο το μέγεθος του index, αφού κάθε σελίδα του νέου index γράφεται ολόκληρη στο WAL. Γι' αυτό η δημιουργία ενός μεγάλου index φαίνεται και ως αιχμή στο lag των replicas του μέρους IV.

Άσκηση 9.2 · Οι εγγραφές ενός UPDATE

Με `pg_walinspect` δείξε τις εγγραφές WAL που γράφει ένα `UPDATE` του `body` της σημείωσης 1 στον πίνακα `notes`, με τον resource manager, τον τύπο και το μήκος τους.

ΛΥΣΗ

```
before=$(psql -At -c "SELECT pg_current_wal_lsn()")
psql -q -c "UPDATE notes SET body = 'αλλαγμένη' WHERE id = 1"
psql -c "SELECT resource_manager, record_type, record_length
         FROM pg_get_wal_records_info('$before', pg_current_wal_lsn())"
```

ΕΞΟΔΟΣ

```
 resource_manager | record_type  | record_length
------------------+--------------+---------------
 XLOG             | FPI_FOR_HINT |           129
 Heap             | HOT_UPDATE   |            87
 Transaction      | COMMIT       |            34
(3 rows)
```

Το `UPDATE` έγραψε μια εγγραφή heap και το commit. Ο τύπος `HOT_UPDATE` σημαίνει ότι η νέα έκδοση μπήκε στην ίδια σελίδα και το index δεν χρειάστηκε αλλαγή, αφού το `body` δεν έχει index. Το `FPI_FOR_HINT` είναι η εικόνα μιας σελίδας λόγω hint bits, μικρή χάρη στο `wal_compression` που ενεργοποιήσαμε.

Άσκηση 9.3 · Τα αρχεία του pg_wal

Δείξε το όνομα του τρέχοντος segment και μέτρησε πόσα αρχεία segments υπάρχουν στο `pg_wal` του `primary`.

ΛΥΣΗ

```
psql -At -c "SELECT pg_walfile_name(pg_current_wal_lsn())"
ls "$LAB/primary/pg_wal" | grep -c '^0000'
```

ΕΞΟΔΟΣ

```
000000010000000000000028
4
```

Τα segments έχουν ονόματα 24 δεκαεξαδικών ψηφίων που αρχίζουν από τον αριθμό του timeline, `00000001`. Το `grep -c` μετρά γραμμές που ταιριάζουν, οπότε αφήνει έξω τον φάκελο `archive_status`. Ο server κρατά μερικά segments έτοιμα για το μέλλον, ανακυκλώνοντας παλιά αντί να φτιάχνει νέα.

Άσκηση 9.4 · Checkpoint με το χέρι

Κάνε `CHECKPOINT` και δείξε από το `pg_stat_checkpointer` πώς άλλαξαν τα `num_requested` και `num_done`.

ΛΥΣΗ

```
SELECT num_requested, num_done FROM pg_stat_checkpointer;
CHECKPOINT;
SELECT pg_stat_clear_snapshot();
SELECT num_requested, num_done FROM pg_stat_checkpointer;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 num_requested | num_done
---------------+----------
             1 |        1
(1 row)

CHECKPOINT

 pg_stat_clear_snapshot
------------------------

(1 row)

 num_requested | num_done
---------------+----------
             2 |        2
(1 row)
```

Και οι δύο μετρητές αυξήθηκαν κατά ένα. Ένα `CHECKPOINT` με το χέρι μετρά ως requested. Το χρησιμοποιείς πριν από ένα `pg_ctl stop` για να γίνει το σταμάτημα πιο γρήγορο, ή πριν από ένα πείραμα όπως αυτά του κεφαλαίου. Το `pg_stat_clear_snapshot` σβήνει την αποθηκευμένη εικόνα των στατιστικών της σύνδεσης, ώστε το δεύτερο `SELECT` να διαβάσει νέες τιμές.

Άσκηση 9.5 · Unlogged σε logged

Μετάτρεψε τον πίνακα `staging` σε κανονικό με `ALTER TABLE ... SET LOGGED`, αφού πρώτα βάλεις 100.000 γραμμές. Μέτρησε πόσο WAL έγραψε η μετατροπή.

ΛΥΣΗ

```
psql -q -c "INSERT INTO staging SELECT n FROM generate_series(1, 100000) AS n"
before=$(psql -At -c "SELECT pg_current_wal_lsn()")
psql -q -c "ALTER TABLE staging SET LOGGED"
psql -At -c "SELECT pg_size_pretty(pg_wal_lsn_diff(pg_current_wal_lsn(), '$before'))"
psql -At -c "SELECT relpersistence FROM pg_class WHERE relname = 'staging'"
```

ΕΞΟΔΟΣ

```
6318 kB
p
```

Η μετατροπή ξαναγράφει ολόκληρο τον πίνακα στο WAL, ώστε να υπάρχει σε recovery και σε replicas. Το `relpersistence` έγινε `p`, permanent. Ένα συνηθισμένο μοτίβο είναι φόρτωση σε unlogged πίνακα, επεξεργασία και μετά `SET LOGGED` μία φορά στο τέλος.

Άσκηση 9.6 · Commit χωρίς αναμονή

Μέσα σε ένα transaction όρισε `synchronous_commit` σε `off` μόνο για αυτό το transaction, δείξε την τιμή και πρόσθεσε ένα event. Δείξε την τιμή ξανά μετά το commit.

ΛΥΣΗ

```
BEGIN;
SET LOCAL synchronous_commit = off;
SHOW synchronous_commit;
INSERT INTO events (customer_id, kind, occurred_at) VALUES (2, 'view', now());
COMMIT;
SHOW synchronous_commit;
```

ΑΠΟΤΕΛΕΣΜΑ

```
BEGIN

SET

 synchronous_commit
--------------------
 off
(1 row)

INSERT 0 1

COMMIT

 synchronous_commit
--------------------
 on
(1 row)
```

Μόνο αυτό το commit επέστρεψε χωρίς να περιμένει τον δίσκο. Τα commits των υπόλοιπων transactions, ακόμη και της ίδιας σύνδεσης, περιμένουν κανονικά.

ΚΕΦΑΛΑΙΟ 10

Autovacuum σε βάθος

Άσκηση 10.1 · Ποιοι πίνακες πλησιάζουν το όριο

Για τους πίνακες του `public` με τουλάχιστον μία νεκρή γραμμή, δείξε το `n_dead_tup` και το ποσοστό του ορίου vacuum που έχουν φτάσει, με ένα δεκαδικό, χρησιμοποιώντας τις ρυθμίσεις του server.

ΛΥΣΗ

```
SELECT relname, n_dead_tup,
       round(100 * n_dead_tup
             / (current_setting('autovacuum_vacuum_threshold')::integer
                + current_setting('autovacuum_vacuum_scale_factor')::numeric * n_live_tup), 1)
         AS pct_of_threshold
FROM pg_stat_user_tables
WHERE schemaname = 'public' AND n_dead_tup > 0
ORDER BY pct_of_threshold DESC;
```

ΑΠΟΤΕΛΕΣΜΑ

```
     relname      | n_dead_tup | pct_of_threshold
------------------+------------+------------------
 pgbench_tellers  |         65 |            108.3
 pgbench_accounts |       3876 |              3.9
(2 rows)
```

Ένα ποσοστό κοντά στο 100 σημαίνει ότι ο autovacuum θα τρέξει σύντομα. Το query αγνοεί τις ρυθμίσεις ανά πίνακα, οπότε για τον `pgbench_accounts` υποτιμά το ποσοστό. Μια πλήρης εκδοχή θα διάβαζε και το `reloptions`.

Άσκηση 10.2 · Ρύθμιση για την ουρά

Φτιάξε πίνακα `job_queue (id bigint PRIMARY KEY, payload text)` με `autovacuum_vacuum_scale_factor` 0 και `autovacuum_vacuum_threshold` 500 κατευθείαν στο `CREATE TABLE`. Δείξε το `reloptions` του.

ΛΥΣΗ

```
CREATE TABLE job_queue (id bigint PRIMARY KEY, payload text)
WITH (autovacuum_vacuum_scale_factor = 0, autovacuum_vacuum_threshold = 500);

SELECT reloptions FROM pg_class WHERE relname = 'job_queue';
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE TABLE

                              reloptions
-----------------------------------------------------------------------
 {autovacuum_vacuum_scale_factor=0,autovacuum_vacuum_threshold=500}
(1 row)
```

Με scale factor μηδέν το όριο δεν εξαρτάται από το μέγεθος. Μια ουρά αδειάζει και γεμίζει συνέχεια, οπότε τρέχει `VACUUM` κάθε 500 νεκρές γραμμές όσο μεγάλη ή μικρή κι αν είναι τη στιγμή εκείνη.

Άσκηση 10.3 · Η παλαιότερη σύνδεση σε transaction

Άνοιξε στο παρασκήνιο ένα `psql` που ξεκινά transaction, διαβάζει και περιμένει τρία δευτερόλεπτα. Όσο περιμένει, δείξε από άλλη σύνδεση την κατάσταση και την ηλικία του `backend_xmin` κάθε σύνδεσης που έχει.

ΛΥΣΗ

```
psql -q -c "BEGIN ISOLATION LEVEL REPEATABLE READ" \
        -c "SELECT count(*) FROM pgbench_branches" \
        -c "SELECT pg_sleep(3)" -c "COMMIT" >/dev/null &
sleep 1
psql -c "SELECT state, age(backend_xmin) AS xmin_age, query
         FROM pg_stat_activity
         WHERE backend_xmin IS NOT NULL AND pid <> pg_backend_pid()"
wait
```

ΕΞΟΔΟΣ

```
 state  | xmin_age |       query
--------+----------+--------------------
 active |        3 | SELECT pg_sleep(3)
(1 row)
```

Το `-c` μπορεί να δοθεί πολλές φορές και το psql εκτελεί τις εντολές με τη σειρά στην ίδια σύνδεση. Η σύνδεση του παρασκηνίου είναι `active` γιατί τρέχει το `pg_sleep`, αλλά κρατά snapshot από το `SELECT` πριν. Το `pid <> pg_backend_pid()` αφήνει έξω τη σύνδεση που κάνει την ερώτηση.

Άσκηση 10.4 · Ηλικία ανά πίνακα

Δείξε τους πέντε πίνακες της βάσης με τη μεγαλύτερη ηλικία `relfrozenxid`, μόνο πίνακες και όχι indexes ή views, μαζί με το μέγεθός τους.

ΛΥΣΗ

```
SELECT oid::regclass AS table_name, age(relfrozenxid) AS xid_age,
       pg_size_pretty(pg_total_relation_size(oid)) AS size
FROM pg_class
WHERE relkind = 'r'
ORDER BY age(relfrozenxid) DESC
LIMIT 5;
```

ΑΠΟΤΕΛΕΣΜΑ

```
      table_name       | xid_age |  size
-----------------------+---------+--------
 pg_statistic_ext_data |    4113 | 16 kB
 pg_authid             |    4113 | 72 kB
 pg_foreign_table      |    4113 | 16 kB
 pg_type               |    4113 | 248 kB
 pg_user_mapping       |    4113 | 24 kB
(5 rows)
```

Μόνο πίνακες έχουν `relfrozenxid`, γι' αυτό το `relkind = 'r'`. Ο μεγάλος πίνακας με μεγάλη ηλικία είναι αυτός που θα πάρει το πιο μακρύ `VACUUM` to prevent wraparound. Σε production θέλεις να το τρέξεις εσύ πρώτος, σε ήσυχη ώρα.

Άσκηση 10.5 · Freeze και visibility map

Τρέξε `VACUUM (FREEZE, VERBOSE)` στον `pgbench_tellers` και εντόπισε στην έξοδο πόσες σελίδες σημειώθηκαν all frozen.

ΛΥΣΗ

```
VACUUM (FREEZE, VERBOSE) pgbench_tellers;
```

ΑΠΟΤΕΛΕΣΜΑ

```
INFO:  aggressively vacuuming "sqlbook.public.pgbench_tellers"
INFO:  finished vacuuming "sqlbook.public.pgbench_tellers": index scans: 0
pages: 0 removed, 1 remain, 1 scanned (100.00% of total), 0 eagerly scanned
tuples: 0 removed, 50 remain, 0 are dead but not yet removable
removable cutoff: 4857, which was 0 XIDs old when operation ended
new relfrozenxid: 4857, which is 191 XIDs ahead of previous value
frozen: 1 pages from table (100.00% of total) had 50 tuples frozen
visibility map: 0 pages set all-visible, 1 pages set all-frozen (1 were all-visible)
index scan not needed: 0 pages from table (0.00% of total) had 0 dead item identifiers removed
avg read rate: 0.000 MB/s, avg write rate: 0.000 MB/s
buffer usage: 23 hits, 0 reads, 0 dirtied
WAL usage: 3 records, 0 full page images, 526 bytes, 0 buffers full
system usage: CPU: user: 0.00 s, system: 0.00 s, elapsed: 0.00 s
VACUUM
```

Η γραμμή `visibility map` λέει πόσες σελίδες σημειώθηκαν all visible και all frozen. Ένα επόμενο `VACUUM` για freeze θα προσπεράσει αυτές τις σελίδες, οπότε σε πίνακες που αλλάζουν λίγο το freeze κοστίζει ελάχιστα μετά την πρώτη φορά. Το `VACUUM` δεν τρέχει μέσα σε transaction, γι' αυτό αυτή η λύση εκτελείται μόνη της.

Άσκηση 10.6 · Autovacuum ανά πίνακα

Δείξε για τους πίνακες του `pgbench` πόσες φορές έτρεξε σε καθέναν autovacuum και autoanalyze και πόσες γραμμές έχουν αλλάξει από το τελευταίο analyze.

ΛΥΣΗ

```
SELECT relname, autovacuum_count, autoanalyze_count, n_mod_since_analyze
FROM pg_stat_user_tables
WHERE relname LIKE 'pgbench%'
ORDER BY relname;
```

ΑΠΟΤΕΛΕΣΜΑ

```
     relname      | autovacuum_count | autoanalyze_count | n_mod_since_analyze
------------------+------------------+-------------------+---------------------
 pgbench_accounts |                1 |                 1 |                4000
 pgbench_branches |                0 |                 1 |                   0
 pgbench_history  |                1 |                 1 |                   0
 pgbench_tellers  |                1 |                 1 |                  50
(4 rows)
```

Το `n_mod_since_analyze` μετρά τις γραμμές που άλλαξαν από το τελευταίο `ANALYZE` και είναι αυτό που συγκρίνεται με το όριο του analyze. Οι πίνακες που άλλαξε το pgbench έχουν περάσει ήδη από autovacuum και autoanalyze μέσα στο κεφάλαιο.

ΚΕΦΑΛΑΙΟ 11

Παρακολούθηση

Άσκηση 11.1 · Τα πιο αργά κατά μέσο όρο

Δείξε από το `pg_stat_statements` τα τρία queries της βάσης με τον μεγαλύτερο μέσο χρόνο εκτέλεσης, από όσα εκτελέστηκαν τουλάχιστον 100 φορές, με τις κλήσεις και τον μέσο χρόνο σε ms με τρία δεκαδικά.

ΛΥΣΗ

```
SELECT left(query, 60) AS query, calls, round(mean_exec_time::numeric, 3) AS mean_ms
FROM pg_stat_statements
WHERE dbid = (SELECT oid FROM pg_database WHERE datname = current_database())
  AND calls >= 100
ORDER BY mean_exec_time DESC
LIMIT 3;
```

ΑΠΟΤΕΛΕΣΜΑ

```
                            query                              | calls | mean_ms
---------------------------------------------------------------+-------+---------
 UPDATE pgbench_branches SET bbalance = bbalance + $1 WHERE b | 75044 |   0.063
 UPDATE pgbench_tellers SET tbalance = tbalance + $1 WHERE ti | 75044 |   0.017
 UPDATE pgbench_accounts SET abalance = abalance + $1 WHERE a | 75044 |   0.009
(3 rows)
```

Το φίλτρο στις κλήσεις αφήνει έξω queries που έτρεξαν μία φορά, όπως το `CREATE EXTENSION`. Ένα query με μεγάλο μέσο χρόνο και λίγες κλήσεις θέλει άλλη αντιμετώπιση από ένα γρήγορο query με εκατομμύρια κλήσεις.

Άσκηση 11.2 · Ένα query, πολλές τιμές

Μηδένισε το `pg_stat_statements`, τρέξε δύο φορές το query που μετρά τους πελάτες μιας πόλης, μία για την πόλη 1 και μία για την πόλη 2. Δείξε μετά πώς καταγράφηκαν.

ΛΥΣΗ

```
SELECT pg_stat_statements_reset() IS NOT NULL AS reset;

SELECT count(*) FROM customers WHERE city_id = 1;
SELECT count(*) FROM customers WHERE city_id = 2;

SELECT query, calls
FROM pg_stat_statements
WHERE query LIKE '%FROM customers WHERE city_id%';
```

ΑΠΟΤΕΛΕΣΜΑ

```
 reset
-------
 t
(1 row)

 count
-------
     7
(1 row)

 count
-------
     6
(1 row)

                        query                        | calls
-----------------------------------------------------+-------
 SELECT count(*) FROM customers WHERE city_id = $1 |     2
(1 row)
```

Τα δύο queries έγιναν μία γραμμή με `$1` και δύο κλήσεις. Η επέκταση δεν κρατά ποιες τιμές χρησιμοποιήθηκαν. Για αυτό χρειάζεσαι το log ή το `auto_explain`.

Άσκηση 11.3 · Μεγάλα sequential scans

Δείξε από το `pg_stat_user_tables` τους τρεις πίνακες με τις περισσότερες γραμμές ανά sequential scan κατά μέσο όρο, μαζί με το πλήθος των sequential scans και το σύνολο των γραμμών που διάβασαν.

ΛΥΣΗ

```
SELECT relname, seq_scan, seq_tup_read,
       seq_tup_read / nullif(seq_scan, 0) AS rows_per_scan
FROM pg_stat_user_tables
ORDER BY rows_per_scan DESC NULLS LAST
LIMIT 3;
```

ΑΠΟΤΕΛΕΣΜΑ

```
     relname      | seq_scan | seq_tup_read | rows_per_scan
------------------+----------+--------------+---------------
 pgbench_accounts |        5 |      1000000 |        200000
 pgbench_tellers  |    75045 |      3752250 |            50
 products         |        4 |           60 |            15
(3 rows)
```

Ο `pgbench_accounts` διαβάζεται ολόκληρος σε κάθε sequential scan, από το query του `auto_explain` που έψαχνε με στήλες χωρίς index. Αν τέτοια scans είναι queries της εφαρμογής, ένα index μάλλον λείπει. Οι μικροί πίνακες του pgbench έχουν χιλιάδες scans αλλά λίγες γραμμές το καθένα. Για πίνακα πέντε γραμμών το sequential scan είναι το φθηνότερο plan και δεν χρειάζεται index. Γι’ αυτό κοιτάς τις γραμμές ανά scan και όχι μόνο το πλήθος των scans.

Άσκηση 11.4 · Πόσα αργά queries

Μέτρησε πόσες γραμμές του log του `primary` είναι για query που ξεπέρασε το `log_min_duration_statement` και δείξε τη μεγαλύτερη διάρκεια.

ΛΥΣΗ

```
grep -c 'duration:' "$LAB/primary.log"
grep -o 'duration: [0-9.]* ms' "$LAB/primary.log" | sort -k2 -n | tail -n 1
```

ΕΞΟΔΟΣ

```
3
duration: 2109.653 ms
```

Το `grep -o` κρατά μόνο το κομμάτι που ταιριάζει και το `sort -k2 -n` ταξινομεί αριθμητικά με το δεύτερο πεδίο. Για πραγματικά logs υπάρχει το `pgBadger`, που φτιάχνει αναφορά HTML από το log της PostgreSQL.

Άσκηση 11.5 · Υγεία της βάσης

Δείξε από το `pg_stat_database` για τη βάση `sqlbook` τα commits, τα rollbacks, τα deadlocks και το ποσοστό των rollbacks στο σύνολο των transactions με δύο δεκαδικά.

ΛΥΣΗ

```
SELECT xact_commit, xact_rollback, deadlocks,
       round(100.0 * xact_rollback / nullif(xact_commit + xact_rollback, 0), 2) AS rollback_pct
FROM pg_stat_database
WHERE datname = 'sqlbook';
```

ΑΠΟΤΕΛΕΣΜΑ

```
 xact_commit | xact_rollback | deadlocks | rollback_pct
-------------+---------------+-----------+--------------
       75278 |             0 |         0 |         0.00
(1 row)
```

Ένα σταθερό ποσοστό rollbacks είναι φυσιολογικό, αφού κάθε σφάλμα εφαρμογής καταλήγει σε rollback. Μια απότομη αύξηση σημαίνει ότι κάτι αρχίζει να αποτυγχάνει μαζικά.

Άσκηση 11.6 · Ποιος γράφει στον δίσκο

Άθροισε από το `pg_stat_io` τις εγγραφές σελίδων σχέσεων ανά είδος διεργασίας και δείξε μόνο όσα έγραψαν κάτι.

ΛΥΣΗ

```
SELECT backend_type, sum(writes) AS writes
FROM pg_stat_io
WHERE object = 'relation'
GROUP BY backend_type
HAVING sum(writes) > 0
ORDER BY writes DESC;
```

ΑΠΟΤΕΛΕΣΜΑ

```
    backend_type    | writes
--------------------+--------
 client backend     |   6212
 checkpointer       |   3092
 standalone backend |   1050
(3 rows)
```

Σε έναν υγιή server τις περισσότερες σελίδες τις γράφει το checkpointer, λίγες ο background writer και σχεδόν καμία τα client backends. Εδώ τα backends έγραψαν κυρίως κατά τη φόρτωση του pgbench, που γεμίζει σελίδες πιο γρήγορα απ' όσο προλαβαίνουν οι βοηθητικές διεργασίες. Ο `standalone backend` είναι το `initdb`, που τρέχει τον server χωρίς postmaster για να φτιάξει το cluster.

ΚΕΦΑΛΑΙΟ 12

pg_dump και pg_restore

Άσκηση 12.1 · Το σχήμα μιας βάσης

Γράψε μόνο το σχήμα όλης της βάσης `sqlbook` σε αρχείο `schema.sql` και μέτρησε πόσες εντολές `CREATE TABLE` περιέχει και σε πόσες γραμμές εμφανίζεται `PRIMARY KEY`.

ΛΥΣΗ

```
pg_dump --schema-only -f schema.sql sqlbook
grep -c '^CREATE TABLE' schema.sql
grep -c 'PRIMARY KEY' schema.sql
```

ΕΞΟΔΟΣ

```
12
12
```

Δώδεκα πίνακες και δώδεκα primary keys. Στο dump τα primary keys δεν είναι μέσα στο `CREATE TABLE`. Γράφονται χωριστά ως `ALTER TABLE ... ADD CONSTRAINT`, στην ενότητα `post-data`, ώστε το index τους να χτιστεί αφού φορτωθούν τα δεδομένα. Προσοχή στο `grep -c`. Όταν δεν βρει τίποτα τυπώνει 0 αλλά επιστρέφει κωδικό αποτυχίας, κάτι που σταματά ένα script με `set -e`.

Άσκηση 12.2 · Πόσα δεδομένα

Μέτρησε από τον πίνακα περιεχομένων του `sqlbook.dump` πόσες εγγραφές είναι δεδομένα πινάκων, πόσες είναι constraints και πόσες foreign keys.

ΛΥΣΗ

```
pg_restore -l sqlbook.dump | grep -c 'TABLE DATA'
pg_restore -l sqlbook.dump | grep -c -E '[0-9] CONSTRAINT'
pg_restore -l sqlbook.dump | grep -c 'FK CONSTRAINT'
```

ΕΞΟΔΟΣ

```
12
18
15
```

Κάθε πίνακας έχει μία εγγραφή δεδομένων. Τα constraints είναι τα primary keys και τα unique, ενώ τα `CHECK` είναι μέρος του `CREATE TABLE`. Τα foreign keys έχουν δικό τους τύπο, `FK CONSTRAINT`. Το pattern `[0-9] CONSTRAINT` ζητά αριθμό αμέσως πριν από τη λέξη, οπότε αφήνει έξω τα foreign keys, όπου μεσολαβεί το `FK`.

Άσκηση 12.3 · Πληρωμές ως INSERT

Εξήγαγε μόνο τα δεδομένα του πίνακα `payments` ως εντολές `INSERT`, μία ανά γραμμή. Δείξε τις τρεις πρώτες.

ΛΥΣΗ

```
pg_dump --data-only --inserts -t payments sqlbook | grep '^INSERT' | head -n 3
```

ΕΞΟΔΟΣ

```
INSERT INTO public.payments VALUES (1, 1, 289.00, 'card', '2025-01-17 20:56:00+02');
INSERT INTO public.payments VALUES (2, 2, 38.00, 'paypal', '2025-01-19 19:04:00+02');
INSERT INTO public.payments VALUES (3, 3, 160.00, 'card', '2025-01-28 16:56:00+02');
```

Το `--inserts` γράφει `INSERT` αντί για `COPY`. Είναι πολύ πιο αργό στο restore, αλλά το αποτέλεσμα φορτώνεται και σε άλλες βάσεις δεδομένων και μπορεί να διαβαστεί γραμμή γραμμή. Το `--column-inserts` προσθέτει και τα ονόματα των στηλών.

Άσκηση 12.4 · Συμπίεση

Φτιάξε dump custom format χωρίς συμπίεση και με συμπίεση `zstd` και σύγκρινε τα μεγέθη με το αρχικό `sqlbook.dump`.

ΛΥΣΗ

```
pg_dump -Fc -Z none -f plain.dump sqlbook
pg_dump -Fc -Z zstd -f zstd.dump sqlbook
ls -l plain.dump sqlbook.dump zstd.dump | awk '{print $5, $9}'
```

ΕΞΟΔΟΣ

```
182794 plain.dump
53070 sqlbook.dump
55717 zstd.dump
```

Το custom format συμπιέζει με gzip από προεπιλογή. Σε μια τόσο μικρή βάση gzip και zstd βγάζουν σχεδόν το ίδιο μέγεθος. Σε dumps πολλών GB το zstd συμπιέζει πολύ πιο γρήγορα για παρόμοιο ή μικρότερο μέγεθος και ο χρόνος του dump μικραίνει αισθητά. Το `awk` κρατά μόνο το μέγεθος και το όνομα από την έξοδο του `ls`.

Άσκηση 12.5 · Μία βάση με άλλο όνομα

Επανάφερε το `sqlbook.dump` στον `restore` σε νέα βάση με όνομα `sqlbook_copy`, χωρίς ιδιοκτήτες. Μέτρησε μετά τους πελάτες της.

ΛΥΣΗ

```
createdb -p 5502 sqlbook_copy
pg_restore -p 5502 -d sqlbook_copy --no-owner sqlbook.dump
psql -p 5502 -d sqlbook_copy -At -c "SELECT count(*) FROM customers"
```

ΕΞΟΔΟΣ

```
30
```

Χωρίς `--create` το `pg_restore` γράφει στη βάση του `-d`, που πρέπει να υπάρχει. Έτσι επαναφέρεις ένα dump με άλλο όνομα, για παράδειγμα για να συγκρίνεις δεδομένα με την κανονική βάση.

Άσκηση 12.6 · Μόνο τα δικαιώματα

Δείξε από τον πίνακα περιεχομένων του `sqlbook.dump` τις εγγραφές δικαιωμάτων και ιδιοκτησίας που αφορούν τον ρόλο `app`, με `pg_restore -l` και `grep`.

ΛΥΣΗ

```
pg_restore -l sqlbook.dump | grep -E 'ACL|app$'
```

ΕΞΟΔΟΣ

```
3982; 0 0 ACL public TABLE customers postgres
234; 1259 16582 TABLE public reviews app
233; 1259 16581 SEQUENCE public reviews_id_seq app
3973; 0 16582 TABLE DATA public reviews app
3988; 0 0 SEQUENCE SET public reviews_id_seq app
3793; 2606 16596 CONSTRAINT public reviews reviews_pkey app
3808; 2606 16602 FK CONSTRAINT public reviews reviews_customer_id_fkey app
3809; 2606 16597 FK CONSTRAINT public reviews reviews_product_id_fkey app
```

Το `GRANT` είναι εγγραφή τύπου `ACL`, access control list. Ο πίνακας `reviews` έχει ιδιοκτήτη τον `app`, που γράφεται στο τέλος κάθε γραμμής της λίστας. Με `--no-privileges` και `--no-owner` το restore τα αγνοεί όλα αυτά.

ΚΕΦΑΛΑΙΟ 13

Physical backup και WAL archiving

Άσκηση 13.1 · Backup σε tar

Πάρε base backup του `primary` σε μορφή tar με συμπίεση gzip στον φάκελο `backup_tar` και δείξε τα αρχεία που δημιουργήθηκαν.

ΛΥΣΗ

```
pg_basebackup -D "$LAB/backup_tar" -Ft -z -X stream -c fast
ls "$LAB/backup_tar"
```

ΕΞΟΔΟΣ

```
backup_manifest
base.tar.gz
pg_wal.tar.gz
```

Το `base.tar.gz` περιέχει τα αρχεία του data directory και το `pg_wal.tar.gz` το WAL της αντιγραφής. Το tar είναι βολικό για μεταφορά και αποθήκευση. Για restore το αποσυμπιέζεις σε έναν άδειο φάκελο και το WAL μέσα στο `pg_wal` του.

Άσκηση 13.2 · Ρόλος για backups

Φτιάξε ρόλο `backup` με `LOGIN` και `REPLICATION`, χωρίς να είναι superuser. Πάρε με αυτόν base backup στον φάκελο `backup_role`. Δείξε τη γραμμή `LABEL` του `backup_label`.

ΛΥΣΗ

```
psql -q -c "CREATE ROLE backup LOGIN REPLICATION"
pg_basebackup -U backup -D "$LAB/backup_role" -X stream -c fast -l "nightly"
grep LABEL "$LAB/backup_role/backup_label"
```

ΕΞΟΔΟΣ

```
LABEL: nightly
```

Το `REPLICATION` αρκεί για το `pg_basebackup`. Ο ρόλος δεν μπορεί να διαβάσει ή να αλλάξει πίνακες με SQL, μόνο να αντιγράψει αρχεία. Το `-l` δίνει ετικέτα στο backup, που γράφεται στο `backup_label`.

Άσκηση 13.3 · Αλλοιωμένο backup

Πρόσθεσε ένα byte στο τέλος του αρχείου `PG_VERSION` του `backup_role` και έλεγξε το backup με `pg_verifybackup`.

ΛΥΣΗ

```
printf 'x' >> "$LAB/backup_role/PG_VERSION"
pg_verifybackup "$LAB/backup_role"
```

ΕΞΟΔΟΣ

```
pg_verifybackup: error: "PG_VERSION" has size 4 on disk but size 3 in the manifest
```

Το μέγεθος του αρχείου δεν ταιριάζει με το manifest, οπότε ο έλεγχος αποτυγχάνει. Ένα backup που δεν ελέγχεις μπορεί να είναι άχρηστο και να το μάθεις τη μέρα που θα το χρειαστείς.

Άσκηση 13.4 · Κατάσταση του archiving

Κλείσε το τρέχον segment με `pg_switch_wal`, περίμενε δύο δευτερόλεπτα και δείξε από το `pg_stat_archiver` αν το τελευταίο αρχειοθετημένο segment είναι αυτό που μόλις έκλεισε.

ΛΥΣΗ

```
pgbench -n -t 10 >/dev/null 2>&1
closed=$(psql -At -c "SELECT pg_walfile_name(pg_switch_wal())")
sleep 2
psql -c "SELECT last_archived_wal = '$closed' AS archived, failed_count FROM pg_stat_archiver"
```

ΕΞΟΔΟΣ

```
 archived | failed_count
----------+--------------
 t        |            5
(1 row)
```

Το `pg_switch_wal` επιστρέφει το LSN όπου τελείωσε το segment και το `pg_walfile_name` το όνομά του. Το pgbench στην αρχή γράφει λίγο WAL. Αν δεν έχει γραφτεί τίποτα από το προηγούμενο switch, το `pg_switch_wal` δεν κλείνει κανένα segment και το όνομα που επιστρέφει είναι του επόμενου, που δεν έχει αρχειοθετηθεί ακόμη. Ένα script παρακολούθησης μπορεί να κάνει τον ίδιο έλεγχο κάθε λίγα λεπτά.

Άσκηση 13.5 · Το archive και το pg_wal

Μέτρησε πόσα αρχεία segments υπάρχουν στο archive και πόσα στο `pg_wal` του `primary`.

ΛΥΣΗ

```
ls "$LAB/archive" | wc -l
ls "$LAB/primary/pg_wal" | grep -c '^0000'
```

ΕΞΟΔΟΣ

```
      12
8
```

Το archive κρατά κάθε segment από τη στιγμή που ενεργοποιήσαμε το archiving, ενώ το `pg_wal` μόνο όσα χρειάζεται ακόμη ο server ή έχει ανακυκλώσει για επόμενη χρήση. Το archive μεγαλώνει για πάντα, εκτός αν σβήνεις ό,τι είναι παλαιότερο από το πιο παλιό backup που θέλεις να κρατήσεις.

Άσκηση 13.6 · Χρόνος ανάμεσα σε segments

Όρισε `archive_timeout` σε 60 δευτερόλεπτα με `ALTER SYSTEM`, κάνε reload και δείξε την τιμή και το context της ρύθμισης.

ΛΥΣΗ

```
psql -q -c "ALTER SYSTEM SET archive_timeout = 60" -c "SELECT pg_reload_conf()"
sleep 1
psql -c "SELECT name, setting, unit, context FROM pg_settings WHERE name = 'archive_timeout'"
```

ΕΞΟΔΟΣ

```
 pg_reload_conf
----------------
 t
(1 row)

      name       | setting | unit | context
-----------------+---------+------+---------
 archive_timeout | 60      | s    | sighup
(1 row)
```

Το `archive_timeout` είναι `sighup`, οπότε αλλάζει με reload. Κάθε segment που κλείνει νωρίς είναι πάντα 16 MB στο archive, αν και μισογεμάτο, οπότε πολύ μικρό `archive_timeout` γεμίζει το archive χωρίς λόγο.

ΚΕΦΑΛΑΙΟ 14

Point in time recovery

Άσκηση 14.1 · Recovery σε restore point

Άλλαξε στον primary όλες τις τιμές των προϊόντων στο διπλάσιο, κλείσε το segment και φτιάξε από το `base` νέο server `rp` στη θύρα 5503 με recovery ως το restore point `before_price_update` και promote στο τέλος. Δείξε την τιμή του προϊόντος 1 στον primary και στον `rp`.

ΛΥΣΗ

```
psql -q -c "UPDATE products SET price = price * 2"
psql -q -c "SELECT pg_switch_wal()"
sleep 2
cp -Rp "$LAB/base" "$LAB/rp"
cat >> "$LAB/rp/postgresql.conf" <<EOF
port = 5503
archive_mode = off
restore_command = 'cp $LAB/archive/%f %p'
recovery_target_name = 'before_price_update'
recovery_target_action = 'promote'
EOF
touch "$LAB/rp/recovery.signal"
pglab start rp
sleep 1
psql -At -p 5501 -c "SELECT price FROM products WHERE id = 1"
psql -At -p 5503 -c "SELECT price FROM products WHERE id = 1"
```

ΕΞΟΔΟΣ

```
 pg_switch_wal
---------------
 0/9016E60
(1 row)

rp: σε λειτουργία στη θύρα 5503
33.80
16.90
```

Ο `rp` σταμάτησε στο restore point, πριν από τον διπλασιασμό. Με `recovery_target_action = 'promote'` έγινε αμέσως κανονικός server, χωρίς παύση. Το restore point φτιάχτηκε μετά το λάθος των πληρωμών, οπότε ο `rp` δεν έχει πληρωμές.

Άσκηση 14.2 · Πού σταμάτησε

Δείξε από το log του `rp` τη γραμμή όπου σταμάτησε το recovery και τη γραμμή με το νέο timeline.

ΛΥΣΗ

```
grep -E 'recovery stopping|selected new timeline' "$LAB/rp.log"
```

ΕΞΟΔΟΣ

```
2026-09-28 09:48:28.734 EEST [78645] LOG:  recovery stopping at restore point
"before_price_update", time 2026-09-28 09:48:25.569355+03
2026-09-28 09:48:28.742 EEST [78645] LOG:  selected new timeline ID: 2
```

Για restore point το log λέει `recovery stopping at restore point` με το όνομα και την ώρα του. Ο `rp` πήρε κι αυτός timeline 2. Είναι ανεξάρτητος από τον `pitr`, που επίσης είναι στο timeline 2, αφού κανείς από τους δύο δεν αρχειοθετεί.

Άσκηση 14.3 · Στοιχεία του pg_control

Με το `pg_controldata` δείξε για τον `rp` την κατάσταση του cluster και το timeline του τελευταίου checkpoint.

ΛΥΣΗ

```
pg_controldata "$LAB/rp" | grep -E "cluster state|Latest checkpoint's TimeLineID"
```

ΕΞΟΔΟΣ

```
Database cluster state:               in production
Latest checkpoint's TimeLineID:       2
```

Το `pg_controldata` διαβάζει το αρχείο `global/pg_control`, την ταυτότητα του cluster. Δουλεύει και όταν ο server είναι σταματημένος, κάτι πολύτιμο όταν διερευνάς γιατί ένας server δεν ξεκινά.

Άσκηση 14.4 · Recovery ως το τέλος του backup

Φτιάξε από το `base` server `early` στη θύρα 5504 με `recovery_target = 'immediate'` και promote και μέτρησε τις γραμμές του `pgbench_history`, όπου κάθε transaction του pgbench γράφει μία γραμμή.

ΛΥΣΗ

```
cp -Rp "$LAB/base" "$LAB/early"
cat >> "$LAB/early/postgresql.conf" <<EOF
port = 5504
archive_mode = off
restore_command = 'cp $LAB/archive/%f %p'
recovery_target = 'immediate'
recovery_target_action = 'promote'
EOF
touch "$LAB/early/recovery.signal"
pglab start early
sleep 1
psql -At -p 5504 -c "SELECT count(*) FROM pgbench_history"
psql -At -p 5501 -c "SELECT count(*) FROM pgbench_history"
```

ΕΞΟΔΟΣ

```
early: σε λειτουργία στη θύρα 5504
0
2000
```

Το `immediate` σταματά μόλις το backup γίνει consistent, όπως στο restore του προηγούμενου κεφαλαίου. Ο `early` δεν έχει κανένα transaction του pgbench, αφού όλα έγιναν μετά το backup. Ο primary έχει 2.000.

Άσκηση 14.5 · Ο pitr μετά το promote

Στον `pitr`, που είναι πλέον κανονικός server, δείξε αν είναι σε recovery και ποια είναι η τρέχουσα θέση του στο WAL μαζί με το όνομα του segment.

ΛΥΣΗ

```
SELECT pg_is_in_recovery() AS in_recovery,
       pg_current_wal_lsn() AS lsn,
       pg_walfile_name(pg_current_wal_lsn()) AS segment;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 in_recovery |    lsn    |         segment
-------------+-----------+--------------------------
 f           | 0/7D32000 | 000000020000000000000007
(1 row)
```

Μετά το promote ο `pitr` γράφει WAL στο timeline 2 και το όνομα του segment αρχίζει από `00000002`. Αν είχε `archive_mode = on`, θα αρχειοθετούσε με αυτά τα ονόματα, χωρίς να συγκρούεται με τα αρχεία του primary.

ΚΕΦΑΛΑΙΟ 15

Incremental backups και έλεγχος των backups

Άσκηση 15.1 · Διαθέσιμες περιλήψεις

Δείξε πόσες περιλήψεις WAL υπάρχουν και από ποιο ως ποιο LSN καλύπτουν συνολικά, από το `pg_available_wal_summaries()`.

ΛΥΣΗ

```
SELECT count(*) AS summaries, min(start_lsn) AS from_lsn, max(end_lsn) AS to_lsn
FROM pg_available_wal_summaries();
```

ΑΠΟΤΕΛΕΣΜΑ

```
 summaries | from_lsn  |  to_lsn
-----------+-----------+-----------
         9 | 0/1000028 | 0/F000028
(1 row)
```

Κάθε περίληψη καλύπτει ένα κομμάτι του WAL. Ένα incremental backup χρειάζεται περιλήψεις χωρίς κενά από το LSN του προηγούμενου backup μέχρι σήμερα. Αν το `summarize_wal` ήταν κλειστό για λίγο ή οι περιλήψεις σβήστηκαν, το incremental αποτυγχάνει και χρειάζεσαι νέο πλήρες.

Άσκηση 15.2 · Πόσο χώρο γλιτώνεις

Δείξε σε KB το μέγεθος του `full`, του `incr1` και του `incr2`. Υπολόγισε μετά με το `awk` τι ποσοστό του πλήρους είναι τα δύο incrementals μαζί.

ΛΥΣΗ

```
du -sk "$LAB/full" "$LAB/incr1" "$LAB/incr2" | awk '{print $1}' > sizes.txt
cat sizes.txt
awk 'NR == 1 {full = $1} NR > 1 {incr += $1} END {printf "%.0f%%\n", 100 * incr / full}' sizes.txt
```

ΕΞΟΔΟΣ

```
202264
51348
38368
44%
```

Τα δύο incrementals μαζί είναι μικρότερα από ένα πλήρες. Με ένα πλήρες την εβδομάδα και έξι incrementals, το σύνολο της εβδομάδας είναι πολύ μικρότερο από επτά πλήρη.

Άσκηση 15.3 · Αλυσίδα χωρίς αρχή

Δοκίμασε να ενώσεις με `pg_combinebackup` μόνο το `incr1` και το `incr2`, χωρίς το πλήρες, σε φάκελο `broken`.

ΛΥΣΗ

```
pg_combinebackup "$LAB/incr1" "$LAB/incr2" -o "$LAB/broken"
```

ΕΞΟΔΟΣ

```
pg_combinebackup: error: backup at "~/pglab/incr1" is an incremental backup, but the
first backup should be a full backup
```

Το εργαλείο αρνείται, γιατί το πρώτο backup της αλυσίδας πρέπει να είναι πλήρες. Ελέγχει επίσης ότι κάθε κρίκος βασίζεται στον προηγούμενο, οπότε δεν μπορείς να ενώσεις backups με λάθος σειρά.

Άσκηση 15.4 · Έλεγχος του primary

Τρέξε `pg_amcheck` σε όλες τις βάσεις του `primary`, με έλεγχο και των γραμμών του heap στα δεδομένα των indexes. Δείξε τον κωδικό εξόδου.

ΛΥΣΗ

```
pg_amcheck --all --install-missing --heapallindexed
echo "exit code: $?"
```

ΕΞΟΔΟΣ

```
exit code: 0
```

Το `--all` ελέγχει κάθε βάση που δέχεται συνδέσεις. Το `--heapallindexed` ελέγχει επιπλέον ότι κάθε γραμμή του heap υπάρχει στα indexes της. Είναι πιο αργό, αλλά πιάνει indexes που έχασαν εγγραφές. Σε μεγάλους servers το τρέχεις σε αντίγραφο και όχι στον primary.

Άσκηση 15.5 · Checksums στο πλήρες backup

Δοκίμασε να ελέγξεις τα checksums του φακέλου `full` με `pg_checksums --check` και δείξε με `pg_controldata` την κατάσταση που γράφει το `pg_control` του.

ΛΥΣΗ

```
pg_controldata "$LAB/full" | grep 'cluster state'
pg_checksums --check -D "$LAB/full"
```

```
ΕΞΟΔΟΣ

Database cluster state:               in production
pg_checksums: error: cluster must be shut down
```

Το `pg_control` του backup αντιγράφηκε όσο ο primary δούλευε, οπότε γράφει `in production`. Το `pg_checksums` δουλεύει μόνο σε cluster που σταμάτησε σωστά. Ένα backup το ελέγχεις αφού το επαναφέρεις, το ξεκινήσεις και το σταματήσεις, όπως κάναμε με το `combined`. Γι' αυτό ο πιο πλήρης έλεγχος ενός backup είναι πάντα ένα restore.

ΚΕΦΑΛΑΙΟ 16

Streaming replication

Άσκηση 16.1 · Από τον primary στον standby

Φτιάξε στον primary πίνακα `rep_test` με 1.000 γραμμές από το `generate_series`, περίμενε μισό δευτερόλεπτο και μέτρησε τις γραμμές στον standby.

ΛΥΣΗ

```
psql -q -c "CREATE TABLE rep_test AS SELECT generate_series(1, 1000) AS n"
sleep 0.5
psql -p 5502 -At -c "SELECT count(*) FROM rep_test"
```

ΕΞΟΔΟΣ

```
1000
```

Ο πίνακας και τα δεδομένα του έφτασαν μαζί, σε μισό δευτερόλεπτο. Και το `CREATE TABLE` γράφει WAL, όπως κάθε αλλαγή.

Άσκηση 16.2 · Τα LSN του standby

Στον standby δείξε το τελευταίο LSN που έλαβε, το τελευταίο που ξαναέπαιξε και αν είναι ίσα.

ΛΥΣΗ

```
SELECT pg_last_wal_receive_lsn() AS received,
       pg_last_wal_replay_lsn() AS replayed,
       pg_last_wal_receive_lsn() = pg_last_wal_replay_lsn() AS caught_up;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 received  | replayed  | caught_up
-----------+-----------+-----------
 0/C7B3C98 | 0/C7B3C98 | t
(1 row)
```

Σε ήσυχο primary τα δύο είναι ίσα. Όταν ο standby λαμβάνει γρηγορότερα απ' όσο ξαναπαίζει, για παράδειγμα επειδή ένα query του καθυστερεί το replay, το `received` προηγείται.

Άσκηση 16.3 · Σύνδεση μόνο σε standby

Με connection string που έχει και τους δύο servers, με τον primary πρώτο, σύνδεσε το psql σε server που είναι σε recovery και δείξε τη θύρα και το `pg_is_in_recovery()`.

ΛΥΣΗ

```
psql "host=localhost,localhost port=5501,5502 target_session_attrs=read-only" \
  -At -c "SELECT current_setting('port'), pg_is_in_recovery()"
```

ΕΞΟΔΟΣ

```
5502|t
```

Το `read-only` δέχεται server που δεν επιτρέπει εγγραφές. Ο standby είναι τέτοιος. Ένας primary με `default_transaction_read_only = on` θα ήταν επίσης. Το `standby` είναι αυστηρότερο και δέχεται μόνο server σε recovery.

Άσκηση 16.4 · Cascading replication

Φτιάξε δεύτερο standby, τον `standby2` στη θύρα 5503, που παίρνει WAL από τον `standby` αντί για τον primary. Δείξε μετά από τον `standby` ποιος συνδέεται σε αυτόν.

ΛΥΣΗ

```
pg_basebackup -p 5502 -D "$LAB/standby2" -R -X stream -c fast
echo "port = 5503" >> "$LAB/standby2/postgresql.conf"
pglab start standby2
sleep 1
psql -p 5502 -c "SELECT application_name, state FROM pg_stat_replication"
```

ΕΞΟΔΟΣ

```
standby2: σε λειτουργία στη θύρα 5503
 application_name |   state
------------------+-----------
 walreceiver      | streaming
(1 row)
```

Ένας standby μπορεί να στείλει WAL σε άλλους standbys. Έτσι ο primary έχει έναν walsender αντί για πολλούς, κάτι χρήσιμο όταν οι standbys είναι σε άλλο datacenter και θέλεις το WAL να ταξιδέψει μία φορά. Το `pg_basebackup` δουλεύει και από standby.

Άσκηση 16.5 · DDL στον standby

Δοκίμασε να φτιάξεις index στον πίνακα `announcements` από τον standby.

ΛΥΣΗ

```
CREATE INDEX ON announcements (body);
```

ΑΠΟΤΕΛΕΣΜΑ

```
ERROR:  cannot execute CREATE INDEX in a read-only transaction
```

Ούτε δεδομένα ούτε σχήμα αλλάζουν στον standby. Ένα index που θέλεις μόνο για τα queries αναφορών του standby πρέπει να φτιαχτεί στον primary και να υπάρχει και εκεί.

ΚΕΦΑΛΑΙΟ 17

Replication slots, lag και conflicts

Άσκηση 17.1 · Πόσο WAL κρατά κάθε slot

Δείξε για κάθε slot το όνομα, αν είναι ενεργό, το `wal_status` και πόσο WAL μπορεί ακόμη να γραφτεί πριν ακυρωθεί, από τη στήλη `safe_wal_size`.

ΛΥΣΗ

```
SELECT slot_name, active, wal_status, pg_size_pretty(safe_wal_size) AS safe
FROM pg_replication_slots;
```

ΑΠΟΤΕΛΕΣΜΑ

```
  slot_name   | active | wal_status |  safe
--------------+--------+------------+--------
 standby_slot | t      | reserved   | 112 MB
(1 row)
```

Το `safe_wal_size` υπάρχει όταν έχει οριστεί `max_slot_wal_keep_size`. Είναι πόσα bytes WAL ακόμη μπορεί να γράψει ο primary πριν το slot χαθεί. Ένα alert όταν πέσει κάτω από ένα όριο δίνει χρόνο να αντιδράσεις.

Άσκηση 17.2 · Slot χωρίς standby

Φτιάξε με `pg_create_physical_replication_slot` ένα slot `spare` που κρατά WAL από τώρα και δείξε το `restart_lsn` και το `active` του. Σβήσε το μετά.

ΛΥΣΗ

```
SELECT slot_name, lsn IS NOT NULL AS reserved
FROM pg_create_physical_replication_slot('spare', true);

SELECT slot_name, active, restart_lsn IS NOT NULL AS has_restart_lsn
FROM pg_replication_slots WHERE slot_name = 'spare';

SELECT pg_drop_replication_slot('spare');
```

ΑΠΟΤΕΛΕΣΜΑ

```
 slot_name | reserved
-----------+----------
 spare     | t
(1 row)

 slot_name | active | has_restart_lsn
-----------+--------+-----------------
 spare     | f      | t
(1 row)

 pg_drop_replication_slot
--------------------------

(1 row)
```

Το δεύτερο όρισμα, `true`, κρατά WAL αμέσως, ώστε ένας standby που θα φτιαχτεί αργότερα από base backup να βρει όλο το WAL. Χωρίς αυτό το slot αρχίζει να κρατά WAL μόνο όταν συνδεθεί κάποιος.

Άσκηση 17.3 · Conflicts ανά αιτία

Δείξε στον standby όλες τις στήλες `confl_` του `pg_stat_database_conflicts` για τη βάση `sqlbook`.

ΛΥΣΗ

```
SELECT confl_tablespace, confl_lock, confl_snapshot, confl_bufferpin,
       confl_deadlock, confl_active_logicalslot
FROM pg_stat_database_conflicts
WHERE datname = 'sqlbook';
```

ΑΠΟΤΕΛΕΣΜΑ

```
 confl_tablespace | confl_lock | confl_snapshot | confl_bufferpin | confl_deadlock | confl_active_logicalslot
------------------+------------+----------------+-----------------+----------------+--------------------------
                0 |          0 |              1 |               0 |              0 |                        0
(1 row)
```

Το `confl_snapshot` είναι το είδος που είδαμε, εκδόσεις γραμμών που σβήστηκαν. Το `confl_lock` προκύπτει όταν ο primary παίρνει `ACCESS EXCLUSIVE` lock, για παράδειγμα με `DROP TABLE` ή `ALTER TABLE`, σε πίνακα που διαβάζει ένα query του standby.

Άσκηση 17.4 · Lag σε bytes για κάθε στάδιο

Δείξε από τον primary για κάθε standby σε bytes πόσο πίσω είναι το write, το flush και το replay από το τρέχον LSN.

ΛΥΣΗ

```
SELECT application_name,
       pg_wal_lsn_diff(pg_current_wal_lsn(), write_lsn) AS write_behind,
       pg_wal_lsn_diff(pg_current_wal_lsn(), flush_lsn) AS flush_behind,
       pg_wal_lsn_diff(pg_current_wal_lsn(), replay_lsn) AS replay_behind
FROM pg_stat_replication;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 application_name | write_behind | flush_behind | replay_behind
------------------+--------------+--------------+---------------
 walreceiver      |            0 |            0 |             0
(1 row)
```

Αν μεγαλώνει μόνο το `replay_behind`, ο standby λαμβάνει κανονικά αλλά κάτι καθυστερεί την εφαρμογή, συνήθως ένα query που προκαλεί conflict και ο standby περιμένει. Αν μεγαλώνουν όλα, το πρόβλημα είναι στο δίκτυο ή στον δίσκο του standby.

Άσκηση 17.5 · Κλείσιμο του feedback

Δείξε στον primary το `xmin` του slot. Κλείσε μετά το `hot_standby_feedback` στον standby και δείξε το ξανά.

ΛΥΣΗ

```
psql -At -c "SELECT xmin IS NOT NULL AS has_xmin FROM pg_replication_slots"
psql -p 5502 -q -c "ALTER SYSTEM SET hot_standby_feedback = off" -c "SELECT pg_reload_conf()"
sleep 2
psql -At -c "SELECT xmin IS NOT NULL AS has_xmin FROM pg_replication_slots"
```

ΕΞΟΔΟΣ

```
t
 pg_reload_conf
----------------
 t
(1 row)

f
```

Με το feedback κλειστό, ο standby στέλνει κενό xmin και ο primary το σβήνει από το slot. Το `VACUUM` του primary δουλεύει χωρίς περιορισμό και τα μακριά queries του standby κινδυνεύουν ξανά.

ΚΕΦΑΛΑΙΟ 18

Synchronous replication

Άσκηση 18.1 · Σειρά προτεραιότητας

Όρισε `synchronous_standby_names` σε `FIRST 1 (standby2, standby1)`, κάνε reload και δείξε για κάθε standby το `sync_priority` και το `sync_state`.

ΛΥΣΗ

```
psql -q -c "ALTER SYSTEM SET synchronous_standby_names = 'FIRST 1 (standby2, standby1)'" \
        -c "SELECT pg_reload_conf()"
sleep 1
psql -c "SELECT application_name, sync_priority, sync_state
         FROM pg_stat_replication ORDER BY sync_priority"
```

ΕΞΟΔΟΣ

```
 pg_reload_conf
----------------
 t
(1 row)

 application_name | sync_priority | sync_state
------------------+---------------+------------
 standby2         |             1 | sync
 standby1         |             2 | potential
(2 rows)
```

Ο `standby2` έχει προτεραιότητα 1 και είναι `sync`. Ο `standby1` έχει 2 και είναι `potential`, δηλαδή θα γίνει `sync` αν χαθεί ο πρώτος. Με `ANY` όλοι έχουν την ίδια προτεραιότητα και είναι `quorum`.

Άσκηση 18.2 · Commit που δεν περιμένει

Σταμάτησε και τους δύο standbys. Κάνε ένα `INSERT` στον `cities` με `synchronous_commit = local` για αυτό το transaction και δείξε ότι τελείωσε. Ξεκίνα μετά τους standbys.

ΛΥΣΗ

```
pglab stop standby1
pglab stop standby2
psql -c "BEGIN" -c "SET LOCAL synchronous_commit = local" \
     -c "INSERT INTO cities VALUES (16, 'Κιλκίς', 'Κεντρική Μακεδονία', 20000)" -c "COMMIT"
pglab start standby1
pglab start standby2
```

ΕΞΟΔΟΣ

```
standby1: σταμάτησε
standby2: σταμάτησε
BEGIN
SET
INSERT 0 1
COMMIT
standby1: σε λειτουργία στη θύρα 5502
standby2: σε λειτουργία στη θύρα 5503
```

Χωρίς standbys ένα κανονικό commit θα κολλούσε. Το `local` το άφησε να τελειώσει μόλις το WAL έφτασε στον δίσκο του primary. Όταν ξεκίνησαν οι standbys, πήραν και αυτή την αλλαγή, απλώς χωρίς εγγύηση για τη στιγμή.

Άσκηση 18.3 · Ο επόμενος στη σειρά

Με το `FIRST 1 (standby2, standby1)` της πρώτης άσκησης, σταμάτησε τον `standby2`, δείξε το `sync_state` του `standby1` και πρόσθεσε μια πόλη. Ξεκίνα μετά ξανά τον `standby2`.

ΛΥΣΗ

```
pglab stop standby2
sleep 1
psql -c "SELECT application_name, sync_state FROM pg_stat_replication"
psql -c "INSERT INTO cities VALUES (17, 'Βέροια', 'Κεντρική Μακεδονία', 66000)"
pglab start standby2
```

ΕΞΟΔΟΣ

```
standby2: σταμάτησε
 application_name | sync_state
------------------+------------
 standby1         | sync
(1 row)

INSERT 0 1
standby2: σε λειτουργία στη θύρα 5503
```

Μόλις χάθηκε ο `standby2`, ο `standby1` πέρασε από `potential` σε `sync` και το `INSERT` τελείωσε κανονικά. Το `FIRST` με δύο ονόματα και ένα `1` δίνει την ίδια ανοχή με το `ANY 1`, με τη διαφορά ότι ξέρεις ποιος standby έχει κάθε φορά την εγγύηση. Αυτό μετράει στο failover, όπου θέλεις να προάγεις τον standby που σίγουρα έχει όλα τα commits.

Άσκηση 18.4 · Πίσω σε asynchronous

Άδειασε το `synchronous_standby_names`, κάνε reload και δείξε το `sync_state` των standbys.

ΛΥΣΗ

```
psql -q -c "ALTER SYSTEM RESET synchronous_standby_names" -c "SELECT pg_reload_conf()"
sleep 1
psql -c "SELECT application_name, sync_state FROM pg_stat_replication ORDER BY 1"
```

ΕΞΟΔΟΣ

```
 pg_reload_conf
----------------
 t
(1 row)

 application_name | sync_state
------------------+------------
 standby1         | async
 standby2         | async
(2 rows)
```

Χωρίς όνομα στη ρύθμιση όλοι οι standbys είναι `async`. Αυτό κάνεις σε ανάγκη, όταν οι synchronous standbys λείπουν και πρέπει να συνεχίσουν οι εγγραφές. Και τα commits που περίμεναν εκείνη τη στιγμή ελευθερώνονται με το reload.

Άσκηση 18.5 · Ορατό στον standby αμέσως

Με `synchronous_standby_names = 'ANY 2 (standby1, standby2)'` και `synchronous_commit = remote_apply` για ένα transaction, πρόσθεσε μια πόλη και διάβασέ την αμέσως από τον `standby2`.

ΛΥΣΗ

```
psql -q -c "ALTER SYSTEM SET synchronous_standby_names = 'ANY 2 (standby1, standby2)'" \
       -c "SELECT pg_reload_conf()"
sleep 1
psql -q -c "BEGIN" -c "SET LOCAL synchronous_commit = remote_apply" \
       -c "INSERT INTO cities VALUES (18, 'Ξάνθη', 'Ανατολική Μακεδονία', 56000)" -c "COMMIT"
psql -p 5503 -At -c "SELECT name FROM cities WHERE id = 18"
```

ΕΞΟΔΟΣ

```
 pg_reload_conf
----------------
 t
(1 row)

Ξάνθη
```

Με `ANY 2` το commit περιμένει και τους δύο standbys να ξαναπαίξουν την αλλαγή, οπότε η ανάγνωση από οποιονδήποτε τη βρίσκει. Με `ANY 1` η εγγύηση θα ίσχυε μόνο για όποιον απάντησε πρώτος.

ΚΕΦΑΛΑΙΟ 19

Failover, timelines και pg_rewind

Άσκηση 19.1 · Timeline στο pg_control

Δείξε με το `pg_controldata` το timeline του τελευταίου checkpoint για τον server της θύρας 5502 και το αρχείο history που έφτιαξε το promote.

ΛΥΣΗ

```
pg_controldata "$LAB/standby" | grep "Latest checkpoint's TimeLineID"
psql -p 5502 -q -c "CHECKPOINT"
pg_controldata "$LAB/standby" | grep "Latest checkpoint's TimeLineID"
cat "$LAB/standby/pg_wal/00000002.history"
```

ΕΞΟΔΟΣ

```
Latest checkpoint's TimeLineID:       1
Latest checkpoint's TimeLineID:       2
1   0/771A9B8   no recovery target specified
```

Μετά το promote ο server δεν κάνει αμέσως checkpoint, για να δεχτεί εγγραφές όσο πιο γρήγορα γίνεται. Μέχρι το πρώτο checkpoint το `pg_control` γράφει ακόμη το timeline 1. Το history λέει ότι το timeline 2 ξεκίνησε από το timeline 1 σε ένα LSN, με λόγο `no recovery target specified`, αφού ήταν απλό promote και όχι recovery με στόχο.

Άσκηση 19.2 · Ποιος είναι primary

Δείξε για τον 5502 αν είναι σε recovery και το timeline του από τα πρώτα οκτώ ψηφία του ονόματος του τρέχοντος segment. Για τον 5501 δείξε αν είναι σε recovery και το `received_tli` του `pg_stat_wal_receiver`.

ΛΥΣΗ

```
psql -p 5502 -At -c "SELECT pg_is_in_recovery(), substr(pg_walfile_name(pg_current_wal_lsn()), 1, 8)"
psql -p 5501 -At -c "SELECT pg_is_in_recovery(), received_tli FROM pg_stat_wal_receiver"
```

ΕΞΟΔΟΣ

```
f|00000002
t|2
```

Ο 5502 είναι primary και γράφει segments του timeline 2. Ο 5501 είναι standby και λαμβάνει WAL του timeline 2. Το `received_tli` είναι το timeline του WAL που έλαβε τελευταίο ο walreceiver.

Άσκηση 19.3 · Switchover

Κάνε switchover ώστε ο server της θύρας 5501 να ξαναγίνει primary. Σταμάτησε κανονικά τον 5502, κάνε promote τον 5501, πρόσθεσε στον 5502 `standby.signal` και `primary_conninfo` προς τον 5501 και ξεκίνα τον. Δείξε το `pg_stat_replication` του 5501.

ΛΥΣΗ

```
pglab stop standby
sleep 1
pg_ctl -D "$LAB/primary" promote
touch "$LAB/standby/standby.signal"
echo "primary_conninfo = 'host=localhost port=5501 user=postgres'" >> "$LAB/standby/postgresql.auto.conf"
pglab start standby
sleep 2
psql -c "SELECT state, sync_state FROM pg_stat_replication"
```

ΕΞΟΔΟΣ

```
standby: σταμάτησε
waiting for server to promote.... done
server promoted
standby: σε λειτουργία στη θύρα 5502
   state   | sync_state
-----------+------------
 streaming | async
(1 row)
```

Ο 5502 σταμάτησε κανονικά και ο 5501 είχε λάβει όλο του το WAL, οπότε μετά το promote ο 5502 ακολουθεί χωρίς `pg_rewind`. Ο 5501 είναι τώρα στο timeline 3. Το `primary_conninfo` στο `postgresql.auto.conf` προστίθεται μετά τη γραμμή που είχε ήδη το αρχείο και ισχύει η τελευταία.

Άσκηση 19.4 · Δοκιμή χωρίς αλλαγές

Σταμάτησε τον 5502 και τρέξε `pg_rewind` με `--dry-run` προς τον 5501. Δείξε τις δύο τελευταίες γραμμές της εξόδου και ξεκίνα ξανά τον 5502.

ΛΥΣΗ

```
pglab stop standby
pg_rewind -D "$LAB/standby" --source-server="host=localhost port=5501 user=postgres dbname=postgres" \
  --dry-run 2>&1 | tail -n 2
pglab start standby
```

ΕΞΟΔΟΣ

```
standby: σταμάτησε
pg_rewind: source and target cluster are on the same timeline
pg_rewind: no rewind required
standby: σε λειτουργία στη θύρα 5502
```

Οι δύο servers έχουν την ίδια ιστορία, οπότε το `pg_rewind` δεν βρίσκει τίποτα να κάνει. Το `--dry-run` δείχνει τι θα γινόταν χωρίς να αλλάξει κανένα αρχείο, κάτι που αξίζει να τρέχεις πριν από κάθε πραγματικό rewind.

Άσκηση 19.5 · Χαμένα commits

Στον 5501 δείξε αν υπάρχει η εγγραφή με id 20 στον `cities` και τι όνομα έχει.

ΛΥΣΗ

```
SELECT id, name FROM cities WHERE id = 20;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id |  name
----+--------
 20 | Καβάλα
(1 row)
```

Η Καβάλα, από το timeline του 5502, πέρασε σε όλα τα επόμενα timelines. Τα Τρίκαλα έζησαν μόνο στην ιστορία του split brain και σβήστηκαν από το `pg_rewind`.

ΚΕΦΑΛΑΙΟ 20

Logical replication

Άσκηση 20.1 · Το slot της subscription

Δείξε στον primary τα slots με το όνομα, τον τύπο, το plugin και αν είναι ενεργά.

ΛΥΣΗ

```
SELECT slot_name, slot_type, plugin, active FROM pg_replication_slots;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 slot_name | slot_type |  plugin  | active
-----------+-----------+----------+--------
 sales_sub | logical   | pgoutput | t
(1 row)
```

Η subscription έφτιαξε ένα logical slot με το όνομά της. Το plugin `pgoutput` μετατρέπει το WAL σε αλλαγές γραμμών για την logical replication. Ό,τι ισχύει για τα physical slots ισχύει κι εδώ. Αν σβήσεις τον subscriber χωρίς `DROP SUBSCRIPTION`, το slot μένει και κρατά WAL.

Άσκηση 20.2 · Νέος πίνακας στην publication

Πρόσθεσε τον `payments` στη `sales_pub`. Φτιάξε τον πίνακα στον `reporting` από dump του σχήματος χωρίς το foreign key, ανανέωσε τη subscription και μέτρησε τις πληρωμές εκεί.

ΛΥΣΗ

```
psql -q -c "ALTER PUBLICATION sales_pub ADD TABLE payments"
pg_dump --schema-only -t payments sqlbook | psql -q -p 5502 >/dev/null 2>&1
psql -p 5502 -q -c "ALTER SUBSCRIPTION sales_sub REFRESH PUBLICATION"
sleep 2
psql -p 5502 -At -c "SELECT count(*) FROM payments"
```

ΕΞΟΔΟΣ

```
155
```

Το foreign key του `payments` προς το `orders` δημιουργήθηκε, αφού το `orders` υπάρχει στον `reporting`. Τα μηνύματα του psql κρύφτηκαν με το `2>&1`, γιατί τα γνωρίζουμε. Η subscription δεν μαθαίνει μόνη της για νέους πίνακες της publication. Το `REFRESH PUBLICATION` ζητά τη νέα λίστα και ξεκινά αρχική αντιγραφή για όσους πίνακες προστέθηκαν.

Άσκηση 20.3 · Στήλη που λείπει

Πρόσθεσε στον primary στήλη `channel text` στον `orders` και μια παραγγελία με `channel = 'web'`. Δείξε το σφάλμα στο log του `reporting`, πρόσθεσε τη στήλη και εκεί και δείξε ότι η παραγγελία έφτασε.

ΛΥΣΗ

```
psql -q -c "ALTER TABLE orders ADD COLUMN channel text"
psql -q -c "INSERT INTO orders (customer_id, status, created_at, channel)
            VALUES (7, 'pending', '2026-07-04 12:00', 'web')"
sleep 2
grep 'missing replicated column' "$LAB/reporting.log" | tail -n 1
psql -p 5502 -q -c "ALTER TABLE orders ADD COLUMN channel text"
sleep 6
psql -p 5502 -At -c "SELECT channel, count(*) FROM orders WHERE channel IS NOT NULL GROUP BY 1"
```

ΕΞΟΔΟΣ

```
2026-09-28 09:50:25.555 EEST [81431] ERROR:  logical replication target relation
"public.orders" is missing replicated column: "channel"
web|1
```

Οι αλλαγές σχήματος δεν αντιγράφονται. Η σωστή σειρά είναι πρώτα η νέα στήλη στον subscriber και μετά στον publisher. Ο subscriber αγνοεί ήσυχα στήλες που έχει και δεν στέλνει ο publisher, αλλά δεν μπορεί να δεχτεί στήλη που δεν έχει.

Άσκηση 20.4 · Παύση της subscription

Κάνε `DISABLE` τη subscription, δείξε στον primary αν το slot είναι ενεργό, κάνε την ξανά `ENABLE` και δείξε το slot ξανά.

ΛΥΣΗ

```
psql -p 5502 -q -c "ALTER SUBSCRIPTION sales_sub DISABLE"
sleep 1
psql -At -c "SELECT slot_name, active FROM pg_replication_slots"
psql -p 5502 -q -c "ALTER SUBSCRIPTION sales_sub ENABLE"
sleep 2
psql -At -c "SELECT slot_name, active FROM pg_replication_slots"
```

ΕΞΟΔΟΣ

```
sales_sub|f
sales_sub|t
```

Με τη subscription σε παύση το slot μένει ανενεργό και ο primary κρατά το WAL από εκεί που σταμάτησε. Όταν ξαναενεργοποιηθεί, η αντιγραφή συνεχίζει χωρίς κενό. Το `DISABLE` είναι χρήσιμο πριν από μια αλλαγή σχήματος στον subscriber.

Άσκηση 20.5 · Κατάσταση της subscription

Στον `reporting` δείξε από το `pg_stat_subscription` τον worker που εφαρμόζει τις αλλαγές, το τελευταίο LSN που έλαβε και αν έχει pid.

ΛΥΣΗ

```
SELECT subname, worker_type, pid IS NOT NULL AS running, received_lsn IS NOT NULL AS receiving
FROM pg_stat_subscription;
```

ΑΠΟΤΕΛΕΣΜΑ

```
  subname  | worker_type | running | receiving
-----------+-------------+---------+-----------
 sales_sub | apply       | t       | t
(1 row)
```

Κάθε subscription έχει έναν `apply` worker. Κατά την αρχική αντιγραφή υπάρχουν και workers τύπου `table synchronization`, ένας ανά πίνακα που αντιγράφεται εκείνη τη στιγμή. Ένας apply worker χωρίς pid σημαίνει ότι η subscription έχει σταματήσει, συνήθως από conflict.

ΚΕΦΑΛΑΙΟ 21

Connection pooling με PgBouncer

Άσκηση 21.1 · Pool σε session mode

Πρόσθεσε στο `pgbouncer.ini` ένα pool `sess` προς την ίδια βάση με `pool_mode = session`, ξαναφόρτωσε τις ρυθμίσεις με `RELOAD` από την κονσόλα και δείξε από το `SHOW DATABASES` το όνομα και το pool mode κάθε βάσης.

ΛΥΣΗ

```
awk '{ print } /^tiny = / { print "sess = host=localhost port=5501 dbname=sqlbook pool_mode=session" }' \
  "$LAB/pgbouncer.ini" > "$LAB/pgbouncer.new"
mv "$LAB/pgbouncer.new" "$LAB/pgbouncer.ini"
psql -p 6501 -d pgbouncer -q -c "RELOAD"
psql -p 6501 -d pgbouncer --csv -c "SHOW DATABASES" |
  awk -F, 'NR == 1 { for (i = 1; i <= NF; i++) if ($i == "pool_mode") c = i } { print $1 "," $c }' |
  column -s, -t
```

ΕΞΟΔΟΣ

```
name       pool_mode
pgbouncer  statement
sess       session
sqlbook
tiny
```

Το κενό `pool_mode` σημαίνει ότι η βάση ακολουθεί την προεπιλογή της ενότητας `[pgbouncer]`, εδώ `transaction`. Το `pool_mode` μπορεί να οριστεί ανά βάση. Έτσι η εφαρμογή χρησιμοποιεί transaction mode και ένα εργαλείο που χρειάζεται session, όπως ένα script migrations με advisory locks, συνδέεται σε άλλο όνομα της ίδιας βάσης. Το `awk` τυπώνει κάθε γραμμή και μετά τη γραμμή του `tiny` προσθέτει τη νέα. Το δεύτερο `awk` βρίσκει στην πρώτη γραμμή σε ποια στήλη είναι το `pool_mode` και τυπώνει αυτή τη στήλη δίπλα στο όνομα.

Άσκηση 21.2 · Η ρύθμιση που δεν διαρρέει

Μέσω του pool `tiny` άλλαξε το `work_mem` σε 64 MB με `SET LOCAL` μέσα σε transaction και δείξε από νέο client του ίδιου pool την τιμή του `work_mem`.

ΛΥΣΗ

```
psql -p 6501 -d tiny -c "RESET work_mem"
psql -p 6501 -d tiny -c "BEGIN" -c "SET LOCAL work_mem = '64MB'" -c "SHOW work_mem" -c "COMMIT"
psql -p 6501 -d tiny -At -c "SHOW work_mem"
```

ΕΞΟΔΟΣ

```
RESET
BEGIN
SET
 work_mem
----------
 64MB
(1 row)

COMMIT
4MB
```

Το πρώτο `RESET` καθαρίζει τη διαρροή του κεφαλαίου. Το `SET LOCAL` ίσχυσε μέσα στο transaction και χάθηκε με το `COMMIT`, οπότε ο επόμενος client βρήκε την προεπιλογή.

Άσκηση 21.3 · Συνδέσεις του PgBouncer στον server

Δείξε από τον server πόσες συνδέσεις έχει κάθε `application_name` και από ποια διεύθυνση, μόνο για client backends.

ΛΥΣΗ

```
SELECT application_name, client_addr, count(*)
FROM pg_stat_activity
WHERE backend_type = 'client backend' AND pid <> pg_backend_pid()
GROUP BY application_name, client_addr
ORDER BY count(*) DESC;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 application_name | client_addr | count
------------------+-------------+-------
 pgbench          | 127.0.0.1   |     3
 pgbench          | ::1         |     2
 psql             | ::1         |     1
(3 rows)
```

Οι συνδέσεις του PgBouncer κρατούν το `application_name` του τελευταίου client που τις χρησιμοποίησε, αφού το PgBouncer το ενημερώνει σε κάθε αλλαγή client. Όλες έρχονται από τη διεύθυνση του PgBouncer.

Άσκηση 21.4 · Παύση για συντήρηση

Κάνε `PAUSE sqlbook` από την κονσόλα, ξεκίνα στο παρασκήνιο ένα query μέσω του PgBouncer, δείξε ότι περιμένει με το `SHOW POOLS` και κάνε `RESUME sqlbook`.

ΛΥΣΗ

```
psql -p 6501 -d pgbouncer -q -c "PAUSE sqlbook"
psql -p 6501 -At -c "SELECT 'πέρασε'" &
sleep 1
psql -p 6501 -d pgbouncer --csv -c "SHOW POOLS" | cut -d, -f 1,3,4 | grep -E 'database|sqlbook' |
  column -s, -t
psql -p 6501 -d pgbouncer -q -c "RESUME sqlbook"
wait
```

ΕΞΟΔΟΣ

```
database  cl_active  cl_waiting
sqlbook   0          1
πέρασε
```

Το `PAUSE` περιμένει να τελειώσουν τα transactions που τρέχουν, κλείνει τις συνδέσεις προς τον server και κρατά τους νέους clients σε αναμονή. Είναι ο τρόπος να κάνεις restart ή switchover του server χωρίς οι clients να δουν σφάλματα, αρκεί η παύση να κρατήσει λίγα δευτερόλεπτα.

Άσκηση 21.5 · Pool που δεν φτάνει

Τρέξε pgbench με 20 clients μέσω του pool `tiny` για δύο δευτερόλεπτα και δείξε τη μέση καθυστέρηση. Κάνε το ίδιο μέσω του `sqlbook`.

ΛΥΣΗ

```
pgbench -n -c 20 -T 2 -p 6501 tiny 2>&1 | grep 'latency average'
pgbench -n -c 20 -T 2 -p 6501 sqlbook 2>&1 | grep 'latency average'
```

ΕΞΟΔΟΣ

```
latency average = 5.844 ms
latency average = 1.977 ms
```

Με μία σύνδεση στον server οι 20 clients περιμένουν σε μία ουρά και η καθυστέρηση κάθε transaction είναι σχεδόν ο χρόνος όλων των άλλων. Με πέντε συνδέσεις η ουρά αδειάζει πολύ πιο γρήγορα. Ένα pool πολύ μικρό για τον φόρτο φαίνεται ως μεγάλο `cl_waiting` και `maxwait` στο `SHOW POOLS`.

ΚΕΦΑΛΑΙΟ 22

Migrations χωρίς downtime

Άσκηση 22.1 · Το lock ενός index

Δείξε με τη `lock_of` ποιο lock παίρνει ένα κανονικό `CREATE INDEX` στο `pgbench_accounts(abalance)`, μέσα σε transaction που αναιρείται.

ΛΥΣΗ

```
CREATE INDEX accounts_abalance_idx ON pgbench_accounts (abalance);
SELECT lock_of('pgbench_accounts') AS lock;
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE INDEX

   lock
___________
 ShareLock
(1 row)
```

Το `ShareLock` επιτρέπει αναγνώσεις αλλά μπλοκάρει `INSERT`, `UPDATE` και `DELETE` για όσο χτίζεται το index. Η λύση εκτελείται σε transaction, γι' αυτό δεν χρειάστηκε `BEGIN`. Το `CONCURRENTLY` δεν θα μπορούσε να δοκιμαστεί έτσι, αφού δεν τρέχει μέσα σε transaction.

Άσκηση 22.2 · Αλλαγή τύπου

Δείξε αν η αλλαγή του `pgbench_accounts.abalance` από `integer` σε `bigint` ξαναγράφει τον πίνακα, συγκρίνοντας το filenode πριν και μέσα στο transaction.

ΛΥΣΗ

```
SELECT pg_relation_filenode('pgbench_accounts') AS before;
ALTER TABLE pgbench_accounts ALTER COLUMN abalance TYPE bigint;
SELECT pg_relation_filenode('pgbench_accounts') AS after, lock_of('pgbench_accounts') AS lock;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 before
--------
  16650
(1 row)

ALTER TABLE

 after |              lock
-------+--------------------------------
 16679 | ShareLock, AccessExclusiveLock
(1 row)
```

Το filenode άλλαξε. Ο `bigint` έχει άλλο μέγεθος στον δίσκο, οπότε κάθε γραμμή ξαναγράφεται με `ACCESS EXCLUSIVE`. Για τέτοια αλλαγή σε μεγάλο πίνακα χρησιμοποιείς expand και contract με νέα στήλη `bigint`.

Άσκηση 22.3 · Μεγαλύτερο varchar

Φτιάξε πίνακα `labels (code varchar(10))` με μία γραμμή και δείξε αν η αλλαγή σε `varchar(50)` και μετά σε `text` αλλάζει το filenode.

ΛΥΣΗ

```
CREATE TABLE labels (code varchar(10));
INSERT INTO labels VALUES ('A-1');
SELECT pg_relation_filenode('labels') AS start;
ALTER TABLE labels ALTER COLUMN code TYPE varchar(50);
SELECT pg_relation_filenode('labels') AS after_varchar50;
ALTER TABLE labels ALTER COLUMN code TYPE text;
SELECT pg_relation_filenode('labels') AS after_text;
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE TABLE

INSERT 0 1

 start
-------
 16683
(1 row)

ALTER TABLE

 after_varchar50
-----------------
           16683
(1 row)

ALTER TABLE

 after_text
------------
      16683
(1 row)
```

Καμία από τις δύο αλλαγές δεν ξαναέγραψε τον πίνακα. Κάθε `varchar(10)` είναι ήδη έγκυρο `varchar(50)` και έγκυρο `text`, με την ίδια αναπαράσταση στον δίσκο. Το αντίθετο, μικρότερο όριο, θέλει έλεγχο κάθε γραμμής.

Άσκηση 22.4 · Χωρίς ουρά

Φτιάξε `CHECK (abalance > -1000000)` στο `pgbench_accounts` ως `NOT VALID`, κάνε το `VALIDATE` και δείξε το `convalidated` πριν και μετά.

ΛΥΣΗ

```
ALTER TABLE pgbench_accounts
  ADD CONSTRAINT abalance_sane CHECK (abalance > -1000000) NOT VALID;
SELECT convalidated FROM pg_constraint WHERE conname = 'abalance_sane';
ALTER TABLE pgbench_accounts VALIDATE CONSTRAINT abalance_sane;
SELECT convalidated FROM pg_constraint WHERE conname = 'abalance_sane';
```

ΑΠΟΤΕΛΕΣΜΑ

```
ALTER TABLE

 convalidated
--------------
 f
(1 row)

ALTER TABLE

 convalidated
--------------
 t
(1 row)
```

Μέχρι το `VALIDATE` το constraint ελέγχει μόνο νέες και αλλαγμένες γραμμές. Αν μια παλιά γραμμή το παραβίαζε, το `VALIDATE` θα αποτύγχανε και το constraint θα έμενε `NOT VALID`, χωρίς να μπλοκάρει κανέναν.

Άσκηση 22.5 · Άκυρα indexes

Βρες όλα τα indexes της βάσης που δεν είναι έγκυρα, με τον πίνακά τους.

ΛΥΣΗ

```
SELECT indexrelid::regclass AS index_name, indrelid::regclass AS table_name
FROM pg_index
WHERE NOT indisvalid;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 index_name | table_name
------------+------------
(0 rows)
```

Το κεφάλαιο έσβησε το άκυρο index που δημιούργησε, οπότε το αποτέλεσμα είναι άδειο. Αυτό το query αξίζει να τρέχει μετά από κάθε deploy, ή ως έλεγχος σε ένα σύστημα παρακολούθησης.

ΚΕΦΑΛΑΙΟ 23

Bloat, REINDEX και pg_repack

Άσκηση 23.1 · Πόσο γεμάτοι είναι οι tellers

Δείξε από το `pgstattuple` το μέγεθος του `pgbench_tellers`, τις ζωντανές και τις νεκρές γραμμές και το ποσοστό ελεύθερου χώρου.

ΛΥΣΗ

```
SELECT table_len, tuple_count, dead_tuple_count, round(free_percent::numeric, 1) AS free_pct
FROM pgstattuple('pgbench_tellers');
```

ΑΠΟΤΕΛΕΣΜΑ

```
 table_len | tuple_count | dead_tuple_count | free_pct
-----------+-------------+------------------+----------
     49152 |         100 |              251 |     62.7
(1 row)
```

Ο πίνακας έχει μερικές σελίδες. Το pgbench αλλάζει συνεχώς τους 100 tellers, οπότε υπάρχουν νεκρές εκδόσεις ανάμεσα σε δύο περάσματα του autovacuum. Σε τόσο μικρό πίνακα το ποσοστό κενού χώρου δεν έχει σημασία.

Άσκηση 23.2 · Index μετά το pg_repack

Δείξε το `avg_leaf_density` και τον αριθμό των φύλλων του `pgbench_accounts_pkey` μετά το `pg_repack`.

ΛΥΣΗ

```
SELECT avg_leaf_density, leaf_pages FROM pgstatindex('pgbench_accounts_pkey');
```

ΑΠΟΤΕΛΕΣΜΑ

```
 avg_leaf_density | leaf_pages
------------------+------------
            91.33 |        438
(1 row)
```

Το `pg_repack` ξαναέχτισε το index από την αρχή, οπότε η πυκνότητα είναι κοντά στο 90 του `fillfactor`. Οι αλλαγές του pgbench στο μεταξύ πρόσθεσαν ελάχιστες σελίδες.

Άσκηση 23.3 · Όλα τα indexes ενός πίνακα

Ξαναχτίσε με `REINDEX TABLE CONCURRENTLY` όλα τα indexes του `pgbench_branches` και δείξε μετά τα indexes του πίνακα με το μέγεθός τους.

ΛΥΣΗ

```
REINDEX TABLE CONCURRENTLY pgbench_branches;

SELECT indexrelid::regclass AS index_name,
       pg_size_pretty(pg_relation_size(indexrelid)) AS size
FROM pg_index WHERE indrelid = 'pgbench_branches'::regclass;
```

ΑΠΟΤΕΛΕΣΜΑ

```
REINDEX

      index_name       | size
-----------------------+-------
 pgbench_branches_pkey | 16 kB
(1 row)
```

Το `REINDEX TABLE CONCURRENTLY` κάνει το ίδιο για κάθε index του πίνακα, ένα προς ένα. Δεν τρέχει μέσα σε transaction, γι' αυτό η λύση εκτελείται μόνη της.

Άσκηση 23.4 · Τι θα έκανε το pg_repack

Τρέξε το `pg_repack` με `--dry-run` για τον πίνακα `accounts_copy` και δείξε την έξοδο.

ΛΥΣΗ

```
pg_repack -d sqlbook -t accounts_copy --dry-run 2>&1
```

ΕΞΟΔΟΣ

```
INFO: Dry run enabled, not executing repack
WARNING: relation "public.accounts_copy" must have a primary key or not-null unique keys
```

Το `accounts_copy` φτιάχτηκε με `CREATE TABLE AS`, οπότε δεν έχει primary key. Το `pg_repack` δεν μπορεί να το ξαναγράψει online, γιατί χρειάζεται ένα μοναδικό κλειδί για να εφαρμόσει τις αλλαγές του log στις σωστές γραμμές. Εδώ θα έμενε μόνο το `VACUUM FULL`.

Άσκηση 23.5 · Μέγεθος χωρίς bloat

Υπολόγισε για το `pgbench_accounts` πόσος χώρος θα χρειαζόταν αν ο πίνακας ήταν γεμάτος, από το `tuple_len` του `pgstattuple`. Σύγκρινέ τον με το πραγματικό μέγεθος.

ΛΥΣΗ

```
SELECT pg_size_pretty(table_len) AS actual,
       pg_size_pretty(tuple_len) AS live_data,
       round(100.0 * tuple_len / table_len, 1) AS live_pct
FROM pgstattuple('pgbench_accounts');
```

ΑΠΟΤΕΛΕΣΜΑ

```
 actual | live_data | live_pct
--------+-----------+----------
 21 MB  | 18 MB     |     88.6
(1 row)
```

Μετά το `pg_repack` ο πίνακας είναι σχεδόν γεμάτος. Το ποσοστό δεν φτάνει ποτέ το 100, γιατί κάθε σελίδα έχει επικεφαλίδα και δείκτες προς τις γραμμές της. Και κάθε γραμμή έχει τη δική της επικεφαλίδα.

ΚΕΦΑΛΑΙΟ 24

Foreign data wrappers

Άσκηση 24.1 · Pushdown ενός LIMIT

Δείξε με `EXPLAIN (VERBOSE, COSTS OFF)` το `Remote SQL` για τα τρία μεγαλύτερα αποθέματα της Θεσσαλονίκης.

ΛΥΣΗ

```
EXPLAIN (VERBOSE, COSTS OFF)
SELECT product_id, qty FROM wh.stock
WHERE center = 'θεσσαλονίκη'
ORDER BY qty DESC
LIMIT 3;
```

ΑΠΟΤΕΛΕΣΜΑ

```
Foreign Scan on wh.stock
  Output: product_id, qty
  Remote SQL: SELECT product_id, qty FROM public.stock WHERE ((center = 'θεσσαλονίκη'))
ORDER BY qty DESC NULLS FIRST LIMIT 3::bigint
```

Το `WHERE`, το `ORDER BY` και το `LIMIT` πήγαν όλα στον `warehouse`, οπότε ταξίδεψαν μόνο τρεις γραμμές. Όταν το `LIMIT` δεν μπορεί να σταλεί, για παράδειγμα επειδή υπάρχει τοπικό φίλτρο, ο τοπικός server πρέπει να φέρει γραμμές μέχρι να βρει τρεις.

Άσκηση 24.2 · Απόθεμα ανά κατηγορία

Δείξε για κάθε κατηγορία το συνολικό απόθεμα στα δύο κέντρα, με join ανάμεσα στα τοπικά `products` και `categories` και στο απομακρυσμένο `wh.stock`.

ΛΥΣΗ

```
SELECT c.name AS category, sum(s.qty) AS units
FROM wh.stock AS s
JOIN products AS p ON p.id = s.product_id
JOIN categories AS c ON c.id = p.category_id
GROUP BY c.name
ORDER BY units DESC, c.name;
```

ΑΠΟΤΕΛΕΣΜΑ

```
     category     | units
------------------+-------
 Laptops          |   127
 Ήχος             |   122
 Περιφερειακά     |   110
 Λογοτεχνία       |   107
 Γραφείο          |    86
 Κουζίνα          |    60
 Βάσεις δεδομένων |    59
 Προγραμματισμός  |    46
 Τεχνολογία       |    13
(9 rows)
```

Το query γράφεται ακριβώς όπως για τοπικούς πίνακες. Η διαφορά φαίνεται μόνο στο plan, όπου ο `stock` διαβάζεται ολόκληρος από τον `warehouse` και το join γίνεται τοπικά.

Άσκηση 24.3 · Νέο κέντρο διανομής

Πρόσθεσε μέσω του foreign table απόθεμα 5 τεμαχίων στο κέντρο `Πάτρα` για τα προϊόντα 1 έως 3 και μέτρησε από τον `warehouse` πόσες γραμμές έχει το νέο κέντρο.

ΛΥΣΗ

```
psql -q -c "INSERT INTO wh.stock SELECT id, 'Πάτρα', 5 FROM products WHERE id <= 3"
psql -p 5502 -d warehouse -At -c "SELECT count(*) FROM stock WHERE center = 'Πάτρα'"
```

ΕΞΟΔΟΣ

```
3
```

Το `INSERT ... SELECT` διάβασε τοπικά τα προϊόντα και έστειλε τις γραμμές στον `warehouse`. Η αλλαγή φαίνεται αμέσως εκεί, αφού έγινε commit μαζί με το τοπικό transaction.

Άσκηση 24.4 · Μέγεθος παρτίδας

Όρισε στον server `warehouse_srv` το `fetch_size` σε 500 και δείξε τις επιλογές του server από το `pg_foreign_server`.

ΛΥΣΗ

```
ALTER SERVER warehouse_srv OPTIONS (ADD fetch_size '500');
SELECT srvname, srvoptions FROM pg_foreign_server WHERE srvname = 'warehouse_srv';
```

ΑΠΟΤΕΛΕΣΜΑ

```
ALTER SERVER

    srvname    |                          srvoptions
---------------+---------------------------------------------------------------
 warehouse_srv | {host=localhost,port=5502,dbname=warehouse,fetch_size=500}
(1 row)
```

Το `postgres_fdw` φέρνει τις γραμμές σε παρτίδες μέσω cursor, 100 από προεπιλογή. Σε μεγάλα αποτελέσματα μια μεγαλύτερη παρτίδα σημαίνει λιγότερα ταξίδια στο δίκτυο. Η επιλογή ορίζεται και ανά foreign table.

Άσκηση 24.5 · Το CSV αλλάζει

Πρόσθεσε μια γραμμή στο `supplier_prices.csv` και μέτρησε ξανά τις γραμμές του foreign table.

ΛΥΣΗ

```
echo "BK-LIT-003,12.10" >> "$LAB/supplier_prices.csv"
psql -At -c "SELECT count(*) FROM supplier_prices"
```

ΕΞΟΔΟΣ

```
5
```

Το `file_fdw` δεν αποθηκεύει τίποτα. Κάθε query ξαναδιαβάζει το αρχείο, οπότε η νέα γραμμή φάνηκε αμέσως. Γι’ αυτό δεν έχει indexes και κάθε query σε μεγάλο αρχείο το διαβάζει ολόκληρο.

ΚΕΦΑΛΑΙΟ 25

pgvector

Άσκηση 25.1 · Τα πιο κοντινά σε ευθεία γραμμή

Βρες τα τρία προϊόντα του `product_vectors` που είναι πιο κοντά στο `[0.0, 1.0, 0.1]` με ευκλείδεια απόσταση, με την απόσταση στρογγυλεμένη σε τρία δεκαδικά.

ΛΥΣΗ

```
SELECT product_id,
       round((features <-> '[0.0, 1.0, 0.1]')::numeric, 3) AS distance
FROM product_vectors
ORDER BY features <-> '[0.0, 1.0, 0.1]'
LIMIT 3;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 product_id | distance
------------+----------
         22 |    0.141
         24 |    0.141
         15 |    1.277
(3 rows)
```

Τα δύο προϊόντα κουζίνας είναι πρώτα. Η ευκλείδεια απόσταση μετράει και το μήκος του vector, οπότε το 22, που έχει βαθμό 1,0 στην κουζίνα, είναι πιο κοντά από το 24 με 0,9. Με απόσταση συνημιτόνου θα μετρούσε μόνο η κατεύθυνση.

Άσκηση 25.2 · Μέσος όρος vectors

Υπολόγισε με το `avg` το μέσο vector των αντικειμένων της κατηγορίας 3 και βρες την απόσταση συνημιτόνου του από το κέντρο της κατηγορίας στον `centers`, με τέσσερα δεκαδικά.

ΛΥΣΗ

```
SELECT round((avg(i.embedding) <=> c.v::vector)::numeric, 4) AS distance
FROM items AS i JOIN centers AS c ON c.category = i.category
WHERE i.category = 3
GROUP BY c.v;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 distance
----------
   0.0005
(1 row)
```

Το `avg` δουλεύει σε vectors διάσταση προς διάσταση. Ο θόρυβος γύρω από το κέντρο είναι συμμετρικός, οπότε ο μέσος όρος 5.000 σημείων πέφτει σχεδόν πάνω του. Ένα μέσο embedding είναι μια απλή περίληψη μιας ομάδας κειμένων.

Άσκηση 25.3 · Χρόνος χωρίς index

Με το index απενεργοποιημένο για ένα transaction, δείξε με `EXPLAIN (ANALYZE, COSTS OFF, BUFFERS OFF)` το plan της αναζήτησης των πέντε κοντινότερων στο αντικείμενο 100.

ΛΥΣΗ

```
SET LOCAL enable_indexscan = off;
EXPLAIN (ANALYZE, COSTS OFF, BUFFERS OFF)
SELECT id FROM items
ORDER BY embedding <=> (SELECT embedding FROM items WHERE id = 100)
LIMIT 5;
```

ΑΠΟΤΕΛΕΣΜΑ

```
SET

Limit (actual time=10.028..10.029 rows=5.00 loops=1)
  InitPlan 1
    ->  Bitmap Heap Scan on items items_1 (actual time=0.006..0.007 rows=1.00 loops=1)
          Recheck Cond: (id = 100)
          Heap Blocks: exact=1
          ->  Bitmap Index Scan on items_pkey (actual time=0.004..0.004 rows=1.00 loops=1)
                Index Cond: (id = 100)
                Index Searches: 1
  ->  Sort (actual time=10.027..10.028 rows=5.00 loops=1)
        Sort Key: ((items.embedding <=> (InitPlan 1).col1))
        Sort Method: top-N heapsort  Memory: 25kB
        ->  Seq Scan on items (actual time=0.012..8.780 rows=50000.00 loops=1)
Planning Time: 0.049 ms
Execution Time: 10.038 ms
```

Χωρίς το index ο planner υπολογίζει την απόσταση για όλα τα 50.000 αντικείμενα και τα ταξινομεί με `top-N heapsort`. Σύγκρινε τον χρόνο με το `Index Scan` του κεφαλαίου. Η διαφορά μεγαλώνει γραμμικά με το μέγεθος του πίνακα.

Άσκηση 25.4 · Index για ευκλείδεια απόσταση

Το index του κεφαλαίου είναι για `<=>`. Δείξε με `EXPLAIN (COSTS OFF)` αν χρησιμοποιείται σε αναζήτηση με `<->`.

ΛΥΣΗ

```
EXPLAIN (COSTS OFF)
SELECT id FROM items
ORDER BY embedding <-> (SELECT embedding FROM items WHERE id = 7)
LIMIT 10;
```

ΑΠΟΤΕΛΕΣΜΑ

```
Limit
  InitPlan 1
    ->  Index Scan using items_pkey on items items_1
          Index Cond: (id = 7)
  ->  Sort
        Sort Key: ((items.embedding <-> (InitPlan 1).col1))
        ->  Seq Scan on items
```

Όχι. Το plan είναι sequential scan με ταξινόμηση. Ένα index `vector_cosine_ops` εξυπηρετεί μόνο τον `<=>`. Για ευκλείδεια απόσταση χρειάζεται index με `vector_l2_ops`. Ο τελεστής του query και η κλάση του index πρέπει να ταιριάζουν, αλλιώς το index αγνοείται σιωπηλά.

Άσκηση 25.5 · Μισό μέγεθος

Δείξε το μέγεθος σε bytes ενός embedding του `items` ως `vector` και ως `halfvec`.

ΛΥΣΗ

```
SELECT pg_column_size(embedding) AS vector_bytes,
       pg_column_size(embedding::halfvec) AS halfvec_bytes
FROM items WHERE id = 1;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 vector_bytes | halfvec_bytes
--------------+---------------
          520 |           264
(1 row)
```

Κάθε διάσταση πιάνει 4 bytes ως `vector` και 2 ως `halfvec`, συν μια μικρή επικεφαλίδα. Για εκατομμύρια embeddings εκατοντάδων διαστάσεων, η διαφορά είναι GB στον δίσκο και στη μνήμη.

ΚΕΦΑΛΑΙΟ 26

PostGIS

Άσκηση 26.1 · Από τη Θεσσαλονίκη στο Ηράκλειο

Υπολόγισε σε km, με ένα δεκαδικό, την απόσταση Θεσσαλονίκης και Ηρακλείου.

ΛΥΣΗ

```
SELECT round(ST_Distance(a.location, b.location)::numeric / 1000, 1) AS km
FROM cities AS a, cities AS b
WHERE a.name = 'Θεσσαλονίκη' AND b.name = 'Ηράκλειο';
```

ΑΠΟΤΕΛΕΣΜΑ

```
  km
-------
 619.3
(1 row)
```

Πάνω από 600 km σε ευθεία, πάνω από το Αιγαίο. Η απόσταση οδικώς ή ακτοπλοϊκώς είναι πολύ μεγαλύτερη. Για διαδρομές σε δρόμους χρειάζεσαι ένα γράφημα δρόμων και την extension pgRouting.

Άσκηση 26.2 · Η πιο μακρινή πόλη

Για κάθε πόλη βρες την πιο μακρινή άλλη πόλη και την απόσταση σε km, χωρίς δεκαδικά, ταξινομημένες κατά απόσταση.

ΛΥΣΗ

```
SELECT a.name, far.name AS farthest, far.km
FROM cities AS a
CROSS JOIN LATERAL (
  SELECT b.name, round(ST_Distance(a.location, b.location)::numeric / 1000) AS km
  FROM cities AS b
  WHERE b.id <> a.id
  ORDER BY km DESC
  LIMIT 1
) AS far
ORDER BY far.km DESC, a.name;
```

ΑΠΟΤΕΛΕΣΜΑ

```
    name     |  farthest   | km
-------------+-------------+-----
 Ιωάννινα    | Ρόδος       | 739
 Ρόδος       | Ιωάννινα    | 739
 Κοζάνη      | Ρόδος       | 707
 Θεσσαλονίκη | Ρόδος       | 655
 Ηράκλειο    | Κοζάνη      | 625
 Λάρισα      | Ρόδος       | 621
 Πάτρα       | Ρόδος       | 608
 Χανιά       | Θεσσαλονίκη | 577
 Βόλος       | Ρόδος       | 566
 Καλαμάτα    | Ρόδος       | 549
 Αθήνα       | Ρόδος       | 434
(11 rows)
```

Το `LATERAL` του πρώτου τόμου τρέχει το subquery μία φορά για κάθε πόλη. Η Ρόδος και τα Ιωάννινα είναι στα δύο άκρα της χώρας και η Ρόδος είναι η πιο μακρινή για τις περισσότερες πόλεις.

Άσκηση 26.3 · Σημεία κοντά στην Πάτρα

Βρες τα τρία σημεία παραλαβής που είναι πιο κοντά στο κέντρο της Πάτρας, με την απόσταση σε μέτρα χωρίς δεκαδικά.

ΛΥΣΗ

```
SELECT p.id, round(ST_Distance(p.location, c.location)) AS meters
FROM pickup_points AS p, (SELECT location FROM cities WHERE name = 'Πάτρα') AS c
ORDER BY p.location <-> c.location
LIMIT 3;
```

ΑΠΟΤΕΛΕΣΜΑ

```
  id   | meters
-------+--------
 83148 |    266
 68849 |    696
 16478 |    821
(3 rows)
```

Το `ORDER BY <->` με `LIMIT` χρησιμοποιεί το GiST index και διαβάζει μόνο λίγους κόμβους, όσο μεγάλος κι αν είναι ο πίνακας.

Άσκηση 26.4 · Πυκνότητα ανά πόλη

Μέτρησε για κάθε πόλη πόσα σημεία παραλαβής απέχουν λιγότερο από 5 km από το κέντρο της και δείξε τις τέσσερις με τα περισσότερα.

ΛΥΣΗ

```
SELECT c.name, count(p.id) AS points
FROM cities AS c
LEFT JOIN pickup_points AS p ON ST_DWithin(c.location, p.location, 5000)
GROUP BY c.name
ORDER BY points DESC, c.name
LIMIT 4;
```

ΑΠΟΤΕΛΕΣΜΑ

```
    name     | points
-------------+--------
 Αθήνα       |   4473
 Θεσσαλονίκη |   1231
 Ηράκλειο    |    211
 Πάτρα       |    155
(4 rows)
```

Τα σημεία είναι απλωμένα ως 20 km από κάθε πόλη, με την τετραγωνική ρίζα στην απόσταση ώστε να καλύπτουν ομοιόμορφα τον κύκλο. Μέσα στα 5 km πέφτει περίπου το ένα δέκατο έκτο των σημείων κάθε πόλης, όσο είναι ο λόγος των εμβαδών.

Άσκηση 26.5 · Σε ποια ζώνη

Για το σημείο παραλαβής 1, δείξε σε ποια ζώνη παράδοσης βρίσκεται, αν βρίσκεται σε κάποια, μαζί με την πόλη του.

ΛΥΣΗ

```
SELECT p.id, p.city, z.name AS zone
FROM pickup_points AS p
LEFT JOIN delivery_zones AS z ON ST_Intersects(z.area, p.location)
WHERE p.id = 1;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id | city  | zone
----+-------+------
  1 | Αθήνα | ∅
(1 row)
```

Το `LEFT JOIN` κρατά το σημείο ακόμη κι αν δεν είναι σε καμία ζώνη, οπότε η στήλη `zone` είναι κενή. Για πολλά σημεία και πολλές ζώνες, ένα GiST index στο `area` των ζωνών κάνει το ίδιο join γρήγορο και από την άλλη πλευρά.