ΣΥΝΟΔΕΥΤΙΚΟΣ ΤΟΜΟΣ

SQL

Λύσεις και εξηγήσεις

Κάθε λύση εκτελέστηκε στην PostgreSQL 18.6 πάνω στα δεδομένα του βιβλίου.

253 **λύσεις σε 41 κεφάλαια**

ΤΟ QUERY, ΤΟ ΑΠΟΤΕΛΕΣΜΑ 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 που αλλάζουν δεδομένα εκτελέστηκαν μέσα σε transaction που έγινε rollback. Κάθε άσκηση ξεκινά λοιπόν από την κατάσταση που έχει η βάση στο τέλος της θεωρίας του κεφαλαίου. Αν έχεις αλλάξει δεδομένα στη δική σου βάση, ξαναφόρτωσέ τη με τον τρόπο του κεφαλαίου 1 πριν συγκρίνεις αποτελέσματα.

Ο ΧΑΡΤΗΣ ΤΟΥ ΒΙΒΛΙΟΥ

Περιεχόμενα

ΚΕΦΑΛΑΙΟ 1

Βάση, πίνακες και το πρώτο SELECT

Άσκηση 1.1 · Όλες οι κατηγορίες

Εμφάνισε όλες τις στήλες του πίνακα `categories`.

ΛΥΣΗ

```
SELECT * FROM categories;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id |       name       | parent_id
----+------------------+-----------
  1 | Βιβλία           |         ∅
  2 | Λογοτεχνία       |         1
  3 | Τεχνολογία       |         1
  4 | Προγραμματισμός  |         3
  5 | Βάσεις δεδομένων |         3
  6 | Ηλεκτρονικά      |         ∅
  7 | Υπολογιστές      |         6
  8 | Laptops          |         7
  9 | Περιφερειακά     |         7
 10 | Ήχος             |         6
 11 | Σπίτι            |         ∅
 12 | Κουζίνα          |        11
 13 | Γραφείο          |        11
(13 rows)
```

Το αστεράκι επιστρέφει τις στήλες με τη σειρά που ορίστηκαν στο `CREATE TABLE`. Η στήλη `parent_id` δείχνει τη γονική κατηγορία και θα την χρησιμοποιήσουμε για να διασχίσουμε το δέντρο στο κεφάλαιο 22.

Άσκηση 1.2 · Κατάλογος προμηθευτών

Εμφάνισε το όνομα, το email και το τηλέφωνο κάθε προμηθευτή από τον πίνακα `suppliers`, με αυτή τη σειρά.

ΛΥΣΗ

```
SELECT name, email, phone FROM suppliers;
```

ΑΠΟΤΕΛΕΣΜΑ

```
         name          |              email               |    phone
-----------------------+----------------------------------+-------------
 Βιβλιοδιανομή Αττικής | orders@vivliodianomi.example.gr | 2105550101
 Techno Import         | sales@technoimport.example.gr   | 2310555202
 Ήχος & Εικόνα ΑΕ      | ∅                                | 2105550303
 Κρητική Οικιακή       | info@kritiki.example.gr          | ∅
 Nordic Office         | hello@nordicoffice.example.com   | +46855500404
 Θεσσαλική Χαρτική     | xartika@example.gr               | 2410555606
(6 rows)
```

Η σειρά των στηλών στο αποτέλεσμα ακολουθεί τη λίστα του `SELECT` και όχι τη σειρά στον πίνακα. Σε δύο γραμμές λείπει τιμή και εμφανίζεται το ∅. Το κεφάλαιο 3 εξηγεί πώς φιλτράρεις τέτοιες γραμμές.

Άσκηση 1.3 · Περιθώριο κέρδους

Για τα πέντε πρώτα προϊόντα εμφάνισε το `name`, την `price`, το `cost` και τη διαφορά τιμής και κόστους με όνομα `margin`.

ΛΥΣΗ

```
SELECT name, price, cost, price - cost AS margin
FROM products
LIMIT 5;
```

ΑΠΟΤΕΛΕΣΜΑ

```
                name                | price | cost  | margin
------------------------------------+-------+-------+--------
 Βίος και πολιτεία του Αλέξη Ζορμπά | 16.90 |  9.10 |   7.80
 Το Κιβώτιο                         | 14.50 |  7.80 |   6.70
 Η Φόνισσα                          |  9.90 |  4.20 |   5.70
 Ματωμένα χώματα                    | 15.20 |  8.00 |   7.20
 Μαθαίνω Python                     | 32.00 | 17.50 |  14.50
(5 rows)
```

Η αφαίρεση γίνεται χωριστά για κάθε γραμμή. Επειδή και οι δύο στήλες είναι `numeric(10,2)`, το αποτέλεσμα κρατά δύο δεκαδικά.

Άσκηση 1.4 · Ονοματεπώνυμο με κόμμα

Εμφάνισε για τους τέσσερις πρώτους πελάτες μία στήλη `customer` με τη μορφή `Επώνυμο, Όνομα` και δίπλα την ημερομηνία γέννησης.

ΛΥΣΗ

```
SELECT last_name || ', ' || first_name AS customer, birth_date
FROM customers
LIMIT 4;
```

ΑΠΟΤΕΛΕΣΜΑ

```
       customer        | birth_date
-----------------------+------------
 Παπαδοπούλου, Μαρία   | 1988-04-12
 Παπαδόπουλος, Γιώργος | 1979-09-30
 Νικολάου, Ελένη       | 1995-01-22
 Δημητρίου, Κώστας     | ∅
(4 rows)
```

Το κόμμα και το κενό είναι ένα ενιαίο κείμενο `', '` ανάμεσα στις δύο στήλες. Ο πελάτης 4 δεν έχει ημερομηνία γέννησης και εμφανίζεται ∅.

Άσκηση 1.5 · Ελληνικές επικεφαλίδες

Εμφάνισε από τον πίνακα `employees` το `full_name` με επικεφαλίδα `Ονοματεπώνυμο` και το `title` με επικεφαλίδα `θέση`, για τους πέντε πρώτους.

ΛΥΣΗ

```
SELECT full_name AS "Ονοματεπώνυμο", title AS "θέση"
FROM employees
LIMIT 5;
```

ΑΠΟΤΕΛΕΣΜΑ

```
     Ονοματεπώνυμο     |        θέση
-----------------------+--------------------
 Ελένη Μαυρίδου        | CEO
 Νίκος Αλεξίου         | CTO
 Σοφία Κωνσταντίνου    | Head of Sales
 Δημήτρης Λαμπρόπουλος | Operations Manager
 Κατερίνα Ζαφειρίου    | Backend Developer
(5 rows)
```

Τα διπλά εισαγωγικά εδώ είναι προαιρετικά, αφού τα ονόματα δεν έχουν κενά. Χωρίς αυτά όμως η PostgreSQL θα τα μετέτρεπε σε πεζά και οι επικεφαλίδες θα γίνονταν `ονοματεπώνυμο` και `θέση`.

Άσκηση 1.6 · Αριθμητική χωρίς πίνακα

Χωρίς `FROM`, υπολόγισε σε ένα query το `17 / 5`, το υπόλοιπο της ίδιας διαίρεσης και το `17.0 / 5`. Δώσε στις στήλες τα ονόματα `quotient`, `remainder` και `exact`.

ΛΥΣΗ

```
SELECT 17 / 5 AS quotient, 17 % 5 AS remainder, 17.0 / 5 AS exact;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 quotient | remainder |       exact
----------+-----------+--------------------
        3 |         2 | 3.4000000000000000
(1 row)
```

Ο αριθμός `17.0` είναι δεκαδικός, οπότε η διαίρεση δεν είναι πια ακέραια. Αρκεί ένας από τους δύο όρους να είναι δεκαδικός. Το κεφάλαιο 5 εξηγεί γιατί το αποτέλεσμα έχει τόσα δεκαδικά ψηφία.

Άσκηση 1.7 · Πόσα χρήματα κάθονται στο ράφι

Για τα τρία πρώτα προϊόντα εμφάνισε το `sku`, το απόθεμα και την αξία του αποθέματος σε τιμή κόστους με όνομα `cost_value`. Στην ίδια γραμμή βάλε και την αξία σε τιμή πώλησης με όνομα `retail_value`.

ΛΥΣΗ

```
SELECT sku, stock, cost * stock AS cost_value, price * stock AS retail_value
FROM products
LIMIT 3;
```

ΑΠΟΤΕΛΕΣΜΑ

```
    sku     | stock | cost_value | retail_value
------------+-------+------------+--------------
 BK-LIT-001 |    42 |     382.20 |       709.80
 BK-LIT-002 |    18 |     140.40 |       261.00
 BK-LIT-003 |    55 |     231.00 |       544.50
(3 rows)
```

Κάθε έκφραση της λίστας υπολογίζεται ανεξάρτητα από τις άλλες πάνω στην ίδια γραμμή. Δεν μπορείς να χρησιμοποιήσεις το alias `cost_value` μέσα σε άλλη έκφραση του ίδιου `SELECT`. Το κεφάλαιο 14 δείχνει πώς το πετυχαίνεις με derived table.

ΚΕΦΑΛΑΙΟ 2

WHERE και λογικές συνθήκες

Άσκηση 2.1 · Ακριβά προϊόντα

Εμφάνισε το όνομα και την τιμή των προϊόντων που κοστίζουν τουλάχιστον 459 ευρώ.

ΛΥΣΗ

```
SELECT name, price
FROM products
WHERE price >= 459;
```

ΑΠΟΤΕΛΕΣΜΑ

```
            name             |  price
-----------------------------+---------
 Laptop Aero 14              | 1149.00
 Laptop Pro 16               | 2199.00
 Laptop Student 15           |  649.00
 Γραφείο ρυθμιζόμενου ύψους |  459.00
 Laptop Classic 13           |  799.00
(5 rows)
```

Το «τουλάχιστον» περιλαμβάνει την τιμή 459, άρα ο τελεστής είναι `>=` και όχι `>`. Με `>` το ρυθμιζόμενο γραφείο θα έλειπε.

Άσκηση 2.2 · Εξαντλημένα αλλά ενεργά

Βρες τα προϊόντα που είναι ενεργά και δεν έχουν απόθεμα. Εμφάνισε `sku`, `name` και `stock`.

ΛΥΣΗ

```
SELECT sku, name, stock
FROM products
WHERE active AND stock = 0;
```

ΑΠΟΤΕΛΕΣΜΑ

```
    sku     |      name       | stock
------------+-----------------+-------
 BK-LIT-004 | Ματωμένα χώματα |     0
 PR-005     | Webcam HD       |     0
(2 rows)
```

Το `Laptop Classic 13` έχει επίσης μηδενικό απόθεμα, αλλά είναι ανενεργό και το `active` το αποκλείει.

Άσκηση 2.3 · Πελάτες της Κρήτης

Οι πόλεις της Κρήτης στον πίνακα `cities` έχουν `id` 4 και 8. Βρες τους πελάτες που μένουν σε αυτές και δεν έχουν εγγραφεί στο newsletter. Εμφάνισε `id`, `first_name`, `last_name` και `city_id`.

ΛΥΣΗ

```
SELECT id, first_name, last_name, city_id
FROM customers
WHERE city_id IN (4, 8) AND NOT newsletter;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id | first_name |  last_name   | city_id
----+------------+--------------+---------
  5 | Αναστασία  | Γεωργίου     |       4
 25 | Λευτέρης   | Φραγκιαδάκης |       8
 29 | Ηλίας      | Τζαννετάκης  |       4
(3 rows)
```

Χωρίς το `IN` η συνθήκη θα ήταν `(city_id = 4 OR city_id = 8) AND NOT newsletter` και οι παρενθέσεις θα ήταν απαραίτητες.

Άσκηση 2.4 · Μεσαία τιμή

Εμφάνισε τα προϊόντα με τιμή από 50 έως 100 ευρώ, με τα δύο άκρα μέσα. Κράτα μόνο όσα ανήκουν στις κατηγορίες 9 ή 10.

ΛΥΣΗ

```
SELECT name, category_id, price
FROM products
WHERE price BETWEEN 50 AND 100
  AND category_id IN (9, 10);
```

ΑΠΟΤΕΛΕΣΜΑ

```
         name          | category_id | price
-----------------------+-------------+-------
 Μηχανικό πληκτρολόγιο |           9 | 89.00
 Webcam HD             |           9 | 59.00
 Ηχείο Bluetooth       |          10 | 79.00
(3 rows)
```

Το `BETWEEN` και το `AND` που ακολουθεί δεν μπερδεύονται. Η PostgreSQL ξέρει ότι το πρώτο `AND` ανήκει στο `BETWEEN`. Για τον αναγνώστη βοηθά η αλλαγή γραμμής.

Άσκηση 2.5 · Email εκτός .gr

Βρες τους προμηθευτές που έχουν email το οποίο δεν τελειώνει σε `.gr`.

ΛΥΣΗ

```
SELECT name, email
FROM suppliers
WHERE email NOT LIKE '%.gr';
```

ΑΠΟΤΕΛΕΣΜΑ

```
     name      |             email
---------------+--------------------------------
 Nordic Office | hello@nordicoffice.example.com
(1 row)
```

Βγαίνει μόνο η `Nordic Office`. Ο προμηθευτής `Ήχος & Εικόνα ΑΕ` δεν έχει email και δεν εμφανίζεται ούτε εδώ ούτε στο αντίθετο query με `LIKE`. Η τιμή που λείπει δεν είναι ούτε `.gr` ούτε κάτι άλλο. Αυτό είναι το θέμα του επόμενου κεφαλαίου.

Άσκηση 2.6 · Laptops που αξίζουν

Εμφάνισε όνομα, τιμή και απόθεμα για τα προϊόντα που το όνομά τους ξεκινά με `Laptop`, είναι ενεργά και κοστίζουν κάτω από 1500 ευρώ ή έχουν απόθεμα πάνω από 5.

ΛΥΣΗ

```
SELECT name, price, stock
FROM products
WHERE name LIKE 'Laptop%'
  AND active
  AND (price < 1500 OR stock > 5);
```

ΑΠΟΤΕΛΕΣΜΑ

```
       name        |  price  | stock
-------------------+---------+-------
 Laptop Aero 14    | 1149.00 |     8
 Laptop Student 15 |  649.00 |    15
(2 rows)
```

Το `Laptop Pro 16` κοστίζει 2199 και έχει απόθεμα 4, οπότε αποτυγχάνει και στα δύο σκέλη του `OR`. Χωρίς παρενθέσεις το `OR stock > 5` θα έφερνε κάθε προϊόν με απόθεμα πάνω από 5, ανεξάρτητα από το όνομα.

Άσκηση 2.7 · Παραγγελίες που δεν παραδόθηκαν

Εμφάνισε τις παραγγελίες του πελάτη 6 που δεν είναι `delivered`. Δείξε `id`, `status` και `coupon_code`.

ΛΥΣΗ

```
SELECT id, status, coupon_code
FROM orders
WHERE customer_id = 6 AND status <> 'delivered';
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id  |  status   | coupon_code
-----+-----------+-------------
  25 | cancelled | WELCOME10
  45 | cancelled | ∅
  55 | refunded  | WELCOME10
 121 | cancelled | ∅
 143 | shipped   | ∅
(5 rows)
```

Το `<>` και το `NOT (status = 'delivered')` είναι ισοδύναμα. Η στήλη `status` δεν επιτρέπει τιμή που λείπει, οπότε εδώ δεν υπάρχει η παγίδα του επόμενου κεφαλαίου.

Άσκηση 2.8 · Αρχή και τέλος ονόματος

Βρες τις πόλεις που το όνομά τους ξεκινά με `Ι` ή `Η` ή τελειώνει σε `ος`, χωρίς διάκριση πεζών και κεφαλαίων. Εμφάνισε όνομα και περιφέρεια.

ΛΥΣΗ

```
SELECT name, region
FROM cities
WHERE name ILIKE 'ι%' OR name ILIKE 'η%' OR name ILIKE '%ος';
```

ΑΠΟΤΕΛΕΣΜΑ

```
   name   |    region
----------+--------------
 Ηράκλειο | Κρήτη
 Βόλος    | Θεσσαλία
 Ιωάννινα | Ήπειρος
 Ρόδος    | Νότιο Αιγαίο
(4 rows)
```

Το `Ηράκλειο` ξεκινά με κεφαλαίο `Η` και το `ILIKE 'η%'` το βρίσκει. Τα `Βόλος` και `Ρόδος` τελειώνουν σε `ος`. Με `LIKE` θα έπρεπε να γράψεις και τα κεφαλαία.

ΚΕΦΑΛΑΙΟ 3

NULL και λογική τριών τιμών

Άσκηση 3.1 · Πελάτες χωρίς τηλέφωνο

Εμφάνισε `id`, `first_name` και `last_name` των πελατών που δεν έχουν τηλέφωνο.

ΛΥΣΗ

```
SELECT id, first_name, last_name
FROM customers
WHERE phone IS NULL;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id | first_name |  last_name
----+------------+--------------
  3 | Ελένη      | Νικολάου
  7 | Δήμητρα    | Κωνσταντίνου
 10 | Χρήστος    | Μιχαηλίδης
 14 | Βασίλης    | Αντωνίου
 17 | Φωτεινή    | Λαμπράκη
 20 | Μιχάλης    | Τσακίρης
 24 | Ζωή        | Παπαγεωργίου
 28 | Δέσποινα   | Μαυρίδου
(8 rows)
```

Με `phone = NULL` το αποτέλεσμα θα ήταν άδειο χωρίς κανένα μήνυμα λάθους.

Άσκηση 3.2 · Κάποιο στοιχείο επικοινωνίας

Βρες τους πελάτες με δηλωμένη πόλη που έχουν τηλέφωνο αλλά όχι email ή email αλλά όχι τηλέφωνο. Εμφάνισε `id`, `email` και `phone`.

ΛΥΣΗ

```
SELECT id, email, phone
FROM customers
WHERE ((email IS NULL AND phone IS NOT NULL)
    OR (email IS NOT NULL AND phone IS NULL))
  AND city_id IS NOT NULL;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id |            email             |   phone
----+------------------------------+------------
  3 | eleni.nik@example.gr         | ∅
  5 | ∅                            | 6955667788
  7 | dimitra.k@example.gr         | ∅
 17 | foteini.l@example.gr         | ∅
 20 | michalis.ts@example.gr       | ∅
 24 | zoi.papageorgiou@example.gr | ∅
(6 rows)
```

Οι εξωτερικές παρενθέσεις κρατούν το `OR` μαζί πριν από το `AND`. Ο πελάτης 14 δεν έχει ούτε email ούτε τηλέφωνο και δεν ταιριάζει σε κανένα σκέλος. Οι πελάτες 10 και 28 έχουν email χωρίς τηλέφωνο, αλλά τους αποκλείει το `city_id IS NOT NULL`.

Άσκηση 3.3 · Εκτός Θεσσαλονίκης

Εμφάνισε `id` και `city_id` όλων των πελατών που δεν είναι σίγουρα από τη Θεσσαλονίκη (πόλη 2), μαζί με όσους δεν έχουν δηλώσει πόλη. Κράτα μόνο πελάτες με `id` από 1 έως 15.

ΛΥΣΗ

```
SELECT id, city_id
FROM customers
WHERE city_id IS DISTINCT FROM 2
  AND id BETWEEN 1 AND 15;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id | city_id
----+---------
  1 |       1
  3 |       1
  4 |       3
  5 |       4
  7 |       5
  8 |       1
  9 |       6
 10 |       ∅
 11 |       7
 12 |       1
 13 |       8
 15 |       9
(12 rows)
```

Ο πελάτης 10 εμφανίζεται με ∅. Με `city_id <> 2` θα χανόταν.

Άσκηση 3.4 · Υπάλληλοι και μισθοί

Εμφάνισε το ονοματεπώνυμο κάθε υπαλλήλου που έχει `id` 11 ή μεγαλύτερο και τον μισθό του. Όπου ο μισθός λείπει, δείξε 0. Ονόμασε τη στήλη `salary_or_zero`.

ΛΥΣΗ

```
SELECT full_name, COALESCE(salary, 0) AS salary_or_zero
FROM employees
WHERE id >= 11;
```

ΑΠΟΤΕΛΕΣΜΑ

```
     full_name      | salary_or_zero
--------------------+----------------
 Βασιλική Ντόκου    |        1100.00
 Γιάννης Κορρές     |        1800.00
 Θανάσης Μπέης      |        1750.00
 Χριστίνα Αθανασίου |        1650.00
 Λευτέρης Γεωργίου  |              0
(5 rows)
```

Το 0 εδώ είναι απόφαση εμφάνισης και όχι δεδομένο. Αν αθροίσεις μισθούς, το `COALESCE` δεν αλλάζει το άθροισμα. Αν όμως υπολογίσεις μέσο όρο, το κεφάλαιο 10 δείχνει ότι η διαφορά μετράει.

Άσκηση 3.5 · Τμήμα με προεπιλογή

Για τους υπαλλήλους 1, 11 και 15 εμφάνισε `full_name` και μια στήλη `label` της μορφής `Τίτλος (Τμήμα)`. Όταν το τμήμα λείπει, γράψε `χωρίς τμήμα` μέσα στην παρένθεση.

ΛΥΣΗ

```
SELECT full_name,
       title || ' (' || COALESCE(department, 'χωρίς τμήμα') || ')' AS label
FROM employees
WHERE id IN (1, 11, 15);
```

ΑΠΟΤΕΛΕΣΜΑ

```
     full_name     |               label
-------------------+-------------------------------------
 Ελένη Μαυρίδου    | CEO (Διοίκηση)
 Βασιλική Ντόκου   | Sales Intern (Πωλήσεις)
 Λευτέρης Γεωργίου | Εξωτερικός σύμβουλος (χωρίς τμήμα)
(3 rows)
```

Χωρίς `COALESCE` ολόκληρη η στήλη `label` του υπαλλήλου 15 θα ήταν ∅, επειδή το `||` με `NULL` δίνει `NULL`.

Άσκηση 3.6 · Παραγγελίες χωρίς κουπόνι WELCOME10

Εμφάνισε `id` και `coupon_code` των παραγγελιών του πελάτη 6 που δεν χρησιμοποίησαν το κουπόνι `WELCOME10`, είτε επειδή είχαν άλλο κουπόνι είτε επειδή δεν είχαν κανένα.

ΛΥΣΗ

```
SELECT id, coupon_code
FROM orders
WHERE customer_id = 6
  AND coupon_code IS DISTINCT FROM 'WELCOME10';
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id  | coupon_code
-----+-------------
   2 | ø
  39 | ø
  42 | ø
  45 | ø
 102 | ø
 121 | ø
 143 | ø
(7 rows)
```

Το `coupon_code <> 'WELCOME10'` θα επέστρεφε μόνο παραγγελίες με άλλο κουπόνι, δηλαδή σχεδόν τίποτα. Οι περισσότερες παραγγελίες δεν έχουν κουπόνι και το `<>` τις πετά.

Άσκηση 3.7 · Περιθώριο χωρίς σφάλμα

Για τα προϊόντα 26 έως 28 υπολόγισε το περιθώριο ως ποσοστό της τιμής, δηλαδή `(price - cost) * 100 / price`, με όνομα `margin_pct`. Δίπλα βάλε στήλη `has_cost` που είναι αληθής όταν το κόστος είναι γνωστό.

ΛΥΣΗ

```
SELECT name,
       (price - cost) * 100 / price AS margin_pct,
       cost IS NOT NULL AS has_cost
FROM products
WHERE id BETWEEN 26 AND 28;
```

ΑΠΟΤΕΛΕΣΜΑ

```
            name             |     margin_pct      | has_cost
-----------------------------+---------------------+----------
 Γραφείο ρυθμιζόμενου ύψους | 34.6405228758169935 | t
 Φωτιστικό γραφείου LED     | 57.1428571428571429 | t
 Δωροκάρτα 50€              |                   ø | f
(3 rows)
```

Το `IS NOT NULL` είναι έκφραση που δίνει boolean, οπότε μπορεί να σταθεί στη λίστα του `SELECT` όπως κάθε άλλη έκφραση. Τα πολλά δεκαδικά του ποσοστού θα τα στρογγυλέψουμε στο κεφάλαιο 5.

ΚΕΦΑΛΑΙΟ 4

ORDER BY, LIMIT και DISTINCT

Άσκηση 4.1 · Τα πέντε ακριβότερα

Εμφάνισε όνομα και τιμή για τα πέντε ακριβότερα ενεργά προϊόντα.

ΛΥΣΗ

```
SELECT name, price
FROM products
WHERE active
ORDER BY price DESC
LIMIT 5;
```

ΑΠΟΤΕΛΕΣΜΑ

```
            name             |  price
-----------------------------+---------
 Laptop Pro 16               | 2199.00
 Laptop Aero 14              | 1149.00
 Laptop Student 15           |  649.00
 Γραφείο ρυθμιζόμενου ύψους |  459.00
 Οθόνη 27" 4K                |  329.00
(5 rows)
```

Το `WHERE active` εκτελείται πριν από την ταξινόμηση, οπότε το ανενεργό `Laptop Classic 13` δεν παίρνει ποτέ θέση στην πεντάδα.

Άσκηση 4.2 · Κατάλογος πελατών

Εμφάνισε επώνυμο και όνομα των πελατών της Αθήνας (πόλη 1), ταξινομημένους κατά επώνυμο και μετά κατά όνομα.

ΛΥΣΗ

```
SELECT last_name, first_name
FROM customers
WHERE city_id = 1
ORDER BY last_name, first_name;
```

ΑΠΟΤΕΛΕΣΜΑ

```
  last_name   | first_name
--------------+------------
 Βασιλείου    | Παναγιώτης
 Βλάχου       | Όλγα
 Ζαχαρίου     | Αικατερίνη
 Κουτσούκος   | Στέλιος
 Νικολάου     | Ελένη
 Οικονόμου    | Αλέξανδρος
 Παπαδοπούλου | Μαρία
(7 rows)
```

Κανένα επώνυμο δεν επαναλαμβάνεται στην Αθήνα, οπότε το δεύτερο κλειδί δεν αλλάζει κάτι εδώ. Αν δύο πελάτες είχαν το ίδιο επώνυμο, θα έκρινε το όνομα. Χωρίς αυτό η σειρά τους θα ήταν απροσδιόριστη.

Άσκηση 4.3 · Οι νεότεροι πελάτες

Εμφάνισε όνομα και ημερομηνία γέννησης των τεσσάρων νεότερων πελατών. Όσοι δεν έχουν δηλώσει ημερομηνία δεν πρέπει να εμφανίζονται.

ΛΥΣΗ

```
SELECT first_name, birth_date
FROM customers
WHERE birth_date IS NOT NULL
ORDER BY birth_date DESC
LIMIT 4;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 first_name | birth_date
------------+------------
 Αικατερίνη | 2003-05-05
 Αναστασία  | 2001-07-08
 Ζωή        | 2000-11-30
 Σοφία      | 1999-12-02
(4 rows)
```

Εδώ το `WHERE birth_date IS NOT NULL` και το `NULLS LAST` δίνουν το ίδιο αποτέλεσμα. Αν όμως οι πελάτες με γνωστή ημερομηνία ήταν λιγότεροι από τέσσερις, το `NULLS LAST` θα γέμιζε τις θέσεις με ∅. Η εκφώνηση λέει ότι δεν πρέπει να εμφανίζονται, άρα το φίλτρο είναι η πιο ακριβής λύση.

Άσκηση 4.4 · Τρίτη σελίδα

Ο κατάλογος εμφανίζει τα προϊόντα από το ακριβότερο στο φθηνότερο, τέσσερα ανά σελίδα. Εμφάνισε `id`, `name` και `price` της τρίτης σελίδας.

ΛΥΣΗ

```
SELECT id, name, price
FROM products
ORDER BY price DESC, id
LIMIT 4 OFFSET 8;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id |         name          | price
----+-----------------------+--------
 30 | Πικάπ                 | 229.00
 19 | Ακουστικά ANC         | 199.00
 17 | USB-C docking station | 139.00
 21 | Μικρόφωνο USB         | 119.00
(4 rows)
```

Η τρίτη σελίδα παραλείπει τις δύο πρώτες, δηλαδή οκτώ γραμμές. Το `id` στο τέλος του `ORDER BY` εγγυάται ότι η σειρά είναι ίδια σε κάθε εκτέλεση.

Άσκηση 4.5 · Περιφέρειες των πελατών

Εμφάνισε χωρίς επαναλήψεις τα ζεύγη `city_id` και `newsletter` των πελατών, ταξινομημένα κατά `city_id` με τα άγνωστα στην αρχή.

ΛΥΣΗ

```
SELECT DISTINCT city_id, newsletter
FROM customers
ORDER BY city_id NULLS FIRST, newsletter;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 city_id | newsletter
---------+------------
       ∅ | f
       1 | f
       1 | t
       2 | f
       2 | t
       3 | f
       4 | f
       4 | t
       5 | f
       5 | t
       6 | t
       7 | f
       7 | t
       8 | f
 …
(17 rows)
```

Το `DISTINCT` εξετάζει το ζεύγος. Η πόλη 1 εμφανίζεται δύο φορές, μία με `false` και μία με `true`. Οι δύο πελάτες χωρίς πόλη έχουν και οι δύο `false`, οπότε δίνουν μία γραμμή.

Άσκηση 4.6 · Ισοβαθμίες μισθών

Εμφάνισε όνομα και μισθό των τριών υπαλλήλων με τον χαμηλότερο γνωστό μισθό. Χρησιμοποίησε `FETCH FIRST` με τρόπο που δεν θα έκοβε ισοβαθμίες, αν υπήρχαν.

ΛΥΣΗ

```
SELECT full_name, salary
FROM employees
WHERE salary IS NOT NULL
ORDER BY salary
FETCH FIRST 3 ROWS WITH TIES;
```

ΑΠΟΤΕΛΕΣΜΑ

```
     full_name      | salary
--------------------+---------
 Βασιλική Ντόκου    | 1100.00
 Χριστίνα Αθανασίου | 1650.00
 Θανάσης Μπέης      | 1750.00
(3 rows)
```

Στα δεδομένα μας δεν υπάρχουν ισοβαθμίες, οπότε το αποτέλεσμα έχει ακριβώς τρεις γραμμές. Το query παραμένει σωστό και την ημέρα που δύο υπάλληλοι θα πάρουν τον ίδιο μισθό.

Άσκηση 4.7 · Η πιο πρόσφατη παραγγελία κάθε πελάτη

Για κάθε έναν από τους πελάτες 1 έως 5 εμφάνισε `customer_id`, το `id` και την ημερομηνία `created_at` της πιο πρόσφατης παραγγελίας του.

ΛΥΣΗ

```
SELECT DISTINCT ON (customer_id) customer_id, id, created_at
FROM orders
WHERE customer_id BETWEEN 1 AND 5
ORDER BY customer_id, created_at DESC;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 customer_id | id  |       created_at
-------------+-----+------------------------
           1 | 129 | 2026-05-26 09:44:00+03
           2 | 128 | 2026-05-25 17:04:00+03
           3 | 145 | 2026-06-14 22:04:00+03
           4 | 141 | 2026-06-09 18:41:00+03
           5 | 137 | 2026-06-06 22:00:00+03
(5 rows)
```

Το `DISTINCT ON (customer_id)` κρατά μία γραμμή ανά πελάτη. Το `created_at DESC` φέρνει πρώτη την πιο πρόσφατη. Το ίδιο πρόβλημα θα το λύσουμε με άλλους δύο τρόπους στα κεφάλαια 14 και 23.

ΚΕΦΑΛΑΙΟ 5

Τύποι, εκφράσεις και CASE

Άσκηση 5.1 · Τιμή με ΦΠΑ

Για τα βιβλία προγραμματισμού (το `sku` ξεκινά με `BK-PRG`) εμφάνισε όνομα, τιμή και τιμή με ΦΠΑ 24% στρογγυλεμένη σε δύο δεκαδικά, με όνομα `with_vat`.

ΛΥΣΗ

```
SELECT name, price, round(price * 1.24, 2) AS with_vat
FROM products
WHERE sku LIKE 'BK-PRG%';
```

ΑΠΟΤΕΛΕΣΜΑ

```
            name             | price | with_vat
-----------------------------+-------+----------
 Μαθαίνω Python              | 32.00 |    39.68
 Ruby από την αρχή           | 29.50 |    36.58
 Καθαρός κώδικας στην πράξη | 38.00 |    47.12
(3 rows)
```

Το `1.24` είναι `numeric` και το `price` επίσης, οπότε ο υπολογισμός είναι ακριβής. Το `round` χρειάζεται επειδή το γινόμενο έχει τέσσερα δεκαδικά.

Άσκηση 5.2 · Domains των προμηθευτών

Εμφάνισε για κάθε προμηθευτή με email το όνομα και το domain του email σε πεζά, με όνομα `domain`. Ταξινόμησε κατά domain.

ΛΥΣΗ

```
SELECT name, lower(split_part(email, '@', 2)) AS domain
FROM suppliers
WHERE email IS NOT NULL
ORDER BY domain;
```

ΑΠΟΤΕΛΕΣΜΑ

```
         name          |          domain
-----------------------+--------------------------
 Θεσσαλική Χαρτική     | example.gr
 Κρητική Οικιακή       | kritiki.example.gr
 Nordic Office         | nordicoffice.example.com
 Techno Import         | technoimport.example.gr
 Βιβλιοδιανομή Αττικής | vivliodianomi.example.gr
(5 rows)
```

Το `ORDER BY domain` χρησιμοποιεί το alias, αφού το `ORDER BY` εκτελείται μετά το `SELECT`. Χωρίς το `WHERE` θα εμφανιζόταν και μια γραμμή με ∅.

Άσκηση 5.3 · Κατηγορίες τιμής

Για τα βιβλία (το `sku` ξεκινά με `BK`) εμφάνισε όνομα, τιμή και στήλη `band` με τιμή `κάτω από 20` για τιμές μικρότερες του 20, `20 έως 40` για τιμές έως και 40 και `πάνω από 40` για τις υπόλοιπες. Ταξινόμησε από το φθηνότερο.

ΛΥΣΗ

```
SELECT name, price,
       CASE
         WHEN price < 20 THEN 'κάτω από 20'
         WHEN price <= 40 THEN '20 έως 40'
         ELSE 'πάνω από 40'
       END AS band
FROM products
WHERE sku LIKE 'BK%'
ORDER BY price;
```

ΑΠΟΤΕΛΕΣΜΑ

```
                name                | price |    band
------------------------------------+-------+-------------
 Η Φόνισσα                          |  9.90 | κάτω από 20
 Το Κιβώτιο                         | 14.50 | κάτω από 20
 Ματωμένα χώματα                    | 15.20 | κάτω από 20
 Βίος και πολιτεία του Αλέξη Ζορμπά | 16.90 | κάτω από 20
 Ruby από την αρχή                  | 29.50 | 20 έως 40
 Μαθαίνω Python                     | 32.00 | 20 έως 40
 Σχεδίαση βάσεων δεδομένων          | 36.00 | 20 έως 40
 Καθαρός κώδικας στην πράξη         | 38.00 | 20 έως 40
 Δίκτυα υπολογιστών                 | 41.00 | πάνω από 40
 PostgreSQL σε βάθος                | 45.00 | πάνω από 40
(10 rows)
```

Η σειρά των `WHEN` είναι μέρος της λογικής. Αν έγραφες πρώτα το `price <= 40`, κάθε φθηνό βιβλίο θα έπεφτε στη μεσαία κατηγορία.

Άσκηση 5.4 · Αρχικά υπαλλήλων

Εμφάνισε για τους υπαλλήλους του τμήματος `Τεχνολογία` το ονοματεπώνυμο και τα αρχικά τους με τελείες, για παράδειγμα `Ν.Α.` για το `Νίκος Αλεξίου`.

ΛΥΣΗ

```
SELECT full_name,
       left(split_part(full_name, ' ', 1), 1) || '.' ||
       left(split_part(full_name, ' ', 2), 1) || '.' AS initials
FROM employees
WHERE department = 'Τεχνολογία'
ORDER BY id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
     full_name      | initials
--------------------+----------
 Νίκος Αλεξίου      | Ν.Α.
 Κατερίνα Ζαφειρίου | Κ.Ζ.
 Άγγελος Φωτίου     | Ά.Φ.
 Μαρίνα Σταθοπούλου | Μ.Σ.
 Πέτρος Χατζής      | Π.Χ.
(5 rows)
```

Ο πίνακας κρατά το όνομα σε μία στήλη, οπότε το `split_part` το χωρίζει στο κενό. Η λύση υποθέτει ότι κάθε ονοματεπώνυμο έχει ακριβώς δύο λέξεις. Γι' αυτό οι πίνακες συνήθως κρατούν όνομα και επώνυμο χωριστά, όπως ο `customers`.

Άσκηση 5.5 · Περιθώριο σε ποσοστό

Για τα προϊόντα της κατηγορίας 10 εμφάνισε όνομα και περιθώριο ως ποσοστό της τιμής, `(price - cost) * 100 / price`, στρογγυλεμένο σε ένα δεκαδικό. Ταξινόμησε από το μεγαλύτερο περιθώριο.

ΛΥΣΗ

```
SELECT name, round((price - cost) * 100 / price, 1) AS margin_pct
FROM products
WHERE category_id = 10
ORDER BY margin_pct DESC;
```

ΑΠΟΤΕΛΕΣΜΑ

```
      name       | margin_pct
-----------------+------------
 Ηχείο Bluetooth |       48.1
 Μικρόφωνο USB   |       46.2
 Ακουστικά ANC   |       39.7
 Πικάπ           |       38.9
(4 rows)
```

Είναι το ίδιο ποσοστό με την άσκηση 3.7. Το `round` εφαρμόζεται στο τελικό αποτέλεσμα. Αν στρογγύλευες κάθε ενδιάμεσο βήμα, το σφάλμα θα συσσωρευόταν.

Άσκηση 5.6 · Ανάλυση SKU

Για τα προϊόντα 11 έως 14 εμφάνισε το `sku`, το πρόθεμά του πριν από την πρώτη παύλα με όνομα `family` και το τελευταίο κομμάτι του ως ακέραιο με όνομα `seq`. Πρόσθεσε στήλη `next_seq` με τον επόμενο αριθμό.

ΛΥΣΗ

```
SELECT sku,
       split_part(sku, '-', 1) AS family,
       split_part(sku, '-', -1)::integer AS seq,
       split_part(sku, '-', -1)::integer + 1 AS next_seq
FROM products
WHERE id BETWEEN 11 AND 14;
```

ΑΠΟΤΕΛΕΣΜΑ

```
  sku   | family | seq | next_seq
--------+--------+-----+----------
 LP-001 | LP     |   1 |        2
 LP-002 | LP     |   2 |        3
 LP-003 | LP     |   3 |        4
 PR-001 | PR     |   1 |        2
(4 rows)
```

Το `split_part(sku, '-', -1)` επιστρέφει το κείμενο `001`. Η μετατροπή σε `integer` το κάνει αριθμό 1 και επιτρέπει την πρόσθεση. Χωρίς αυτήν το `'001' + 1` θα έδινε σφάλμα, επειδή το `split_part` επιστρέφει `text`.

Άσκηση 5.7 · Ετικέτα καταλόγου

Εμφάνισε για τα προϊόντα της κατηγορίας 12 μία στήλη `label` της μορφής `Βραστήρας · 34.90 € · σε απόθεμα`. Όταν το απόθεμα είναι κάτω από 10 γράψε `λίγα τεμάχια` αντί για `σε απόθεμα`.

ΛΥΣΗ

```
SELECT format('%s · %s € · %s',
              name,
              to_char(price, 'FM9990.00'),
              CASE WHEN stock < 10 THEN 'λίγα τεμάχια' ELSE 'σε απόθεμα' END) AS label
FROM products
WHERE category_id = 12
ORDER BY id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
                    label
---------------------------------------------
 Καφετιέρα espresso · 249.00 € · λίγα τεμάχια
 Βραστήρας · 34.90 € · σε απόθεμα
 Σετ μαχαιριών · 64.00 € · σε απόθεμα
(3 rows)
```

Το `CASE` είναι μία από τις τιμές που περνούν στο `format`. Οποιαδήποτε έκφραση μπορεί να σταθεί εκεί.

ΚΕΦΑΛΑΙΟ 6

Ημερομηνίες, ώρες και intervals

Άσκηση 6.1 · Παραγγελίες του Φεβρουαρίου

Εμφάνισε `id`, `customer_id` και `created_at` των παραγγελιών του Φεβρουαρίου 2026, από την πιο πρόσφατη.

ΛΥΣΗ

```
SELECT id, customer_id, created_at
FROM orders
WHERE created_at >= '2026-02-01' AND created_at < '2026-03-01'
ORDER BY created_at DESC;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id | customer_id |       created_at
----+-------------+------------------------
 92 |          11 | 2026-02-23 21:54:00+02
 91 |           1 | 2026-02-23 21:07:00+02
 90 |           5 | 2026-02-23 14:48:00+02
 89 |           2 | 2026-02-23 09:41:00+02
 88 |           9 | 2026-02-22 12:45:00+02
 87 |           5 | 2026-02-19 14:30:00+02
 86 |          19 | 2026-02-14 12:23:00+02
 85 |          28 | 2026-02-14 11:22:00+02
 84 |           9 | 2026-02-12 16:22:00+02
 83 |           5 | 2026-02-05 20:54:00+02
 82 |          13 | 2026-02-04 09:09:00+02
 81 |          21 | 2026-02-02 08:04:00+02
(12 rows)
```

Το άνω όριο είναι η 1η Μαρτίου με `<`. Έτσι δεν χρειάζεται να ξέρεις αν ο Φεβρουάριος έχει 28 ή 29 ημέρες.

Άσκηση 6.2 · Αργές αποστολές

Βρες τις παραγγελίες του 2026 που χρειάστηκαν πάνω από τέσσερις ημέρες για να αποσταλούν. Εμφάνισε `id`, την ημερομηνία παραγγελίας ως `date` και τον χρόνο αποστολής με όνομα `shipping_time`.

ΛΥΣΗ

```
SELECT id, created_at::date AS ordered_on, shipped_at - created_at AS shipping_time
FROM orders
WHERE created_at >= '2026-01-01'
  AND shipped_at - created_at > interval '4 days'
ORDER BY id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id  | ordered_on |  shipping_time
-----+------------+-----------------
  76 | 2026-01-10 | 4 days 04:00:00
  79 | 2026-01-26 | 4 days 01:00:00
  82 | 2026-02-04 | 4 days 06:00:00
  85 | 2026-02-14 | 4 days 04:00:00
  90 | 2026-02-23 | 4 days 06:00:00
  93 | 2026-03-01 | 4 days 01:00:00
  94 | 2026-03-04 | 4 days 02:00:00
  95 | 2026-03-06 | 4 days 01:00:00
  98 | 2026-03-16 | 4 days 01:00:00
 102 | 2026-03-25 | 4 days 01:00:00
 104 | 2026-03-31 | 4 days 03:00:00
 112 | 2026-04-23 | 4 days 05:00:00
 132 | 2026-05-30 | 4 days 01:00:00
 136 | 2026-06-05 | 4 days 05:00:00
 …
(21 rows)
```

Η σύγκριση γίνεται ανάμεσα σε δύο `interval`. Οι παραγγελίες χωρίς `shipped_at` δίνουν `NULL` στη διαφορά και φεύγουν από το `WHERE`, που εδώ είναι το σωστό.

Άσκηση 6.3 · Παραγγελίες του Σαββατοκύριακου

Εμφάνισε `id` και `created_at` των παραγγελιών του Μαΐου 2026 που έγιναν Σάββατο ή Κυριακή, με όνομα ημέρας στα αγγλικά σε στήλη `day`.

ΛΥΣΗ

```
SELECT id, created_at, to_char(created_at, 'Day') AS day
FROM orders
WHERE created_at >= '2026-05-01' AND created_at < '2026-06-01'
  AND extract(isodow FROM created_at) IN (6, 7)
ORDER BY created_at;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id  |       created_at       |    day
-----+------------------------+-----------
 117 | 2026-05-09 15:51:00+03 | Saturday
 121 | 2026-05-17 15:39:00+03 | Sunday
 125 | 2026-05-23 23:07:00+03 | Saturday
 126 | 2026-05-24 18:43:00+03 | Sunday
 132 | 2026-05-30 16:20:00+03 | Saturday
 133 | 2026-05-31 19:38:00+03 | Sunday
(6 rows)
```

Το `extract` υπολογίζει την ημέρα στη ζώνη της σύνδεσης. Μια παραγγελία στις 01:00 ώρα Ελλάδας το Σάββατο είναι ακόμη Παρασκευή σε UTC. Γι' αυτό η ζώνη της σύνδεσης είναι μέρος της ερώτησης.

Άσκηση 6.4 · Ηλικίες

Για τους πελάτες που γεννήθηκαν πριν από το 1980 εμφάνισε όνομα, ημερομηνία γέννησης και την ηλικία τους σε ολόκληρα έτη στις 30 Ιουνίου 2026, με όνομα `years`.

ΛΥΣΗ

```
SELECT first_name, birth_date,
       extract(year FROM age(date '2026-06-30', birth_date)) AS years
FROM customers
WHERE birth_date < '1980-01-01'
ORDER BY birth_date;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 first_name | birth_date | years
------------+------------+-------
 Βασίλης    | 1965-10-10 |    60
 Παναγιώτης | 1972-05-27 |    54
 Λευτέρης   | 1975-04-04 |    51
 Σπύρος     | 1976-12-24 |    49
 Γιώργος    | 1979-09-30 |    46
 Giorgos    | 1979-09-30 |    46
(6 rows)
```

Το `age` επιστρέφει `interval` με έτη, μήνες και ημέρες. Το `extract(year ...)` κρατά μόνο τα έτη, δηλαδή τα ολοκληρωμένα χρόνια. Η διαίρεση των ημερών με 365 θα έκανε λάθος γύρω από τα γενέθλια.

Άσκηση 6.5 · Μήνας και εβδομάδα

Για τις παραγγελίες του πελάτη 13 εμφάνισε `id`, τον μήνα σε μορφή `YYYY-MM` με όνομα `month` και την ημερομηνία της Δευτέρας της εβδομάδας με όνομα `week_start`.

ΛΥΣΗ

```
SELECT id,
       to_char(created_at, 'YYYY-MM') AS month,
       date_trunc('week', created_at)::date AS week_start
FROM orders
WHERE customer_id = 13
ORDER BY created_at;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id |  month  | week_start
----+---------+------------
 20 | 2025-05 | 2025-04-28
 35 | 2025-09 | 2025-09-01
 40 | 2025-09 | 2025-09-15
 41 | 2025-09 | 2025-09-15
 82 | 2026-02 | 2026-02-02
(5 rows)
```

Το `date_trunc('week', ...)` δίνει τα μεσάνυχτα της Δευτέρας ως `timestamptz`. Η μετατροπή σε `date` κρατά μόνο την ημέρα.

Άσκηση 6.6 · Οι Δευτέρες του Ιουνίου

Εμφάνισε με `generate_series` όλες τις Δευτέρες του Ιουνίου 2026 ως `date`, σε στήλη `monday`.

ΛΥΣΗ

```
SELECT d::date AS monday
FROM generate_series(date '2026-06-01', date '2026-06-30', interval '1 day') AS d
WHERE extract(isodow FROM d) = 1;
```

ΑΠΟΤΕΛΕΣΜΑ

```
   monday
------------
 2026-06-01
 2026-06-08
 2026-06-15
 2026-06-22
 2026-06-29
(5 rows)
```

Η σειρά παράγει όλες τις ημέρες και το `WHERE` κρατά τις Δευτέρες. Αφού η 1η Ιουνίου 2026 είναι Δευτέρα, θα αρκούσε και βήμα `interval '7 days'` χωρίς φίλτρο. Η λύση με φίλτρο δουλεύει για κάθε μήνα.

Άσκηση 6.7 · Η ίδια στιγμή σε τρεις πόλεις

Για την παραγγελία 100 εμφάνισε το `created_at` και την ώρα τοίχου της ίδιας στιγμής στο Λονδίνο και στο Τόκιο, σε στήλες `london` και `tokyo`.

ΛΥΣΗ

```
SELECT created_at,
       created_at AT TIME ZONE 'Europe/London' AS london,
       created_at AT TIME ZONE 'Asia/Tokyo' AS tokyo
FROM orders
WHERE id = 100;
```

ΑΠΟΤΕΛΕΣΜΑ

```
       created_at       |       london        |        tokyo
------------------------+---------------------+---------------------
 2026-03-22 13:04:00+02 | 2026-03-22 11:04:00 | 2026-03-22 20:04:00
(1 row)
```

Οι στήλες `london` και `tokyo` είναι `timestamp` χωρίς ζώνη. Είναι ώρες τοίχου και όχι στιγμές. Αν τις συγκρίνεις μεταξύ τους, θα συγκρίνεις ενδείξεις ρολογιών και όχι χρόνο.

Άσκηση 6.8 · Πρόσφατες προσλήψεις

Εμφάνισε όνομα και ημερομηνία πρόσληψης των υπαλλήλων που προσλήφθηκαν τα δύο τελευταία χρόνια πριν από τις 30 Ιουνίου 2026, μαζί με τις ημέρες που έχουν περάσει από την πρόσληψη, σε στήλη `days`.

ΛΥΣΗ

```
SELECT full_name, hired_on, date '2026-06-30' - hired_on AS days
FROM employees
WHERE hired_on >= date '2026-06-30' - interval '2 years'
ORDER BY hired_on;
```

ΑΠΟΤΕΛΕΣΜΑ

```
     full_name     |  hired_on  | days
-------------------+------------+------
 Πέτρος Χατζής     | 2024-09-02 |  666
 Λευτέρης Γεωργίου | 2025-01-07 |  539
 Βασιλική Ντόκου   | 2025-10-01 |  272
(3 rows)
```

Το `date - interval` δίνει `timestamp`, που συγκρίνεται κανονικά με τη `hired_on`. Η διαφορά δύο `date` είναι ακέραιος αριθμός ημερών.

ΚΕΦΑΛΑΙΟ 7

Κλειδιά και INNER JOIN

Άσκηση 7.1 · Πελάτες του 2026 και πόλεις

Εμφάνισε όνομα, επώνυμο και όνομα πόλης των πελατών που εγγράφηκαν το 2026, ταξινομημένους κατά ημερομηνία εγγραφής.

ΛΥΣΗ

```
SELECT c.first_name, c.last_name, t.name AS city
FROM customers AS c
JOIN cities AS t ON t.id = c.city_id
WHERE c.created_at >= '2026-01-01'
ORDER BY c.created_at;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 first_name |  last_name  |    city
------------+-------------+-------------
 Πέτρος     | Σαββίδης    | Θεσσαλονίκη
 Ηλίας      | Τζαννετάκης | Ηράκλειο
 Όλγα       | Βλάχου      | Αθήνα
(3 rows)
```

Ο πελάτης 28 εγγράφηκε τον Φεβρουάριο του 2026 αλλά δεν έχει πόλη, οπότε το `INNER JOIN` τον αφήνει έξω. Αν η ερώτηση ήταν «όλοι οι νέοι πελάτες», θα χρειαζόσουν το `LEFT JOIN` του επόμενου κεφαλαίου.

Άσκηση 7.2 · Περιφερειακά

Εμφάνισε όνομα και τιμή των προϊόντων της κατηγορίας με όνομα `Περιφερειακά`, χωρίς να γράψεις τον αριθμό της κατηγορίας.

ΛΥΣΗ

```
SELECT p.name, p.price
FROM products AS p
JOIN categories AS cat ON cat.id = p.category_id
WHERE cat.name = 'Περιφερειακά'
ORDER BY p.price;
```

ΑΠΟΤΕΛΕΣΜΑ

```
         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 `WHERE` φιλτράρει με στήλη του δεύτερου πίνακα. Μετά το `JOIN` όλες οι στήλες και των δύο πινάκων είναι διαθέσιμες σε κάθε επόμενο κομμάτι του query.

Άσκηση 7.3 · Λογαριασμός παραγγελίας

Για την παραγγελία 144 εμφάνισε το όνομα κάθε προϊόντος, την ποσότητα, την τιμή μονάδας και το σύνολο της γραμμής με όνομα `line_total`.

ΛΥΣΗ

```
SELECT p.name, oi.quantity, oi.unit_price,
       oi.quantity * oi.unit_price AS line_total
FROM order_items AS oi
JOIN products AS p ON p.id = oi.product_id
WHERE oi.order_id = 144
ORDER BY p.name;
```

ΑΠΟΤΕΛΕΣΜΑ

```
       name       | quantity | unit_price | line_total
------------------+----------+------------+------------
 Ασύρματο ποντίκι |        1 |      24.90 |      24.90
 Ματωμένα χώματα  |        2 |      15.20 |      30.40
 Σετ μαχαιριών    |        2 |      64.00 |     128.00
(3 rows)
```

Το σύνολο υπολογίζεται με την `unit_price` της παραγγελίας και όχι με την τρέχουσα `products.price`. Για παλιές παραγγελίες οι δύο μπορεί να διαφέρουν.

Άσκηση 7.4 · Παραγγελίες από την Κρήτη

Εμφάνισε `id` παραγγελίας, όνομα πελάτη και πόλη κατοικίας για τις παραγγελίες του Μαΐου 2026 από πελάτες που μένουν στην περιφέρεια Κρήτης.

ΛΥΣΗ

```
SELECT o.id, c.first_name, c.last_name, t.name AS city
FROM orders AS o
JOIN customers AS c ON c.id = o.customer_id
JOIN cities AS t ON t.id = c.city_id
WHERE t.region = 'Κρήτη'
  AND o.created_at >= '2026-05-01' AND o.created_at < '2026-06-01'
ORDER BY o.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id  | first_name |  last_name   |   city
-----+------------+--------------+----------
 115 | Ηλίας      | Τζαννετάκης  | Ηράκλειο
 123 | Ηλίας      | Τζαννετάκης  | Ηράκλειο
 127 | Λευτέρης   | Φραγκιαδάκης | Χανιά
 131 | Ηλίας      | Τζαννετάκης  | Ηράκλειο
(4 rows)
```

Το φίλτρο της περιφέρειας εφαρμόζεται στον τρίτο πίνακα της αλυσίδας. Δεν χρειάζεται να ξέρεις τα `id` των πόλεων της Κρήτης.

Άσκηση 7.5 · Προμηθευτές ακουστικών

Εμφάνισε το όνομα κάθε προμηθευτή που φέρνει το προϊόν `Ακουστικά ANC` και την τιμή προμήθειας.

ΛΥΣΗ

```
SELECT s.name AS supplier, ps.supply_price
FROM products AS p
JOIN product_suppliers AS ps ON ps.product_id = p.id
JOIN suppliers AS s ON s.id = ps.supplier_id
WHERE p.name = 'Ακουστικά ANC'
ORDER BY ps.supply_price;
```

ΑΠΟΤΕΛΕΣΜΑ

```
     supplier     | supply_price
------------------+--------------
 Ήχος & Εικόνα ΑΕ |       120.00
 Techno Import    |       124.80
(2 rows)
```

Ο `product_suppliers` συνδέει προϊόντα και προμηθευτές σε σχέση πολλά προς πολλά. Ένα προϊόν έχει πολλούς προμηθευτές και ένας προμηθευτής πολλά προϊόντα. Για να περάσεις από τη μία πλευρά στην άλλη χρειάζονται δύο joins.

Άσκηση 7.6 · Πληρωμές επιστροφών

Εμφάνισε `id` παραγγελίας, ποσό, μέθοδο και ημερομηνία πληρωμής για όλες τις πληρωμές των παραγγελιών με κατάσταση `refunded`.

ΛΥΣΗ

```
SELECT o.id, pay.amount, pay.method, pay.paid_at::date AS paid_on
FROM orders AS o
JOIN payments AS pay ON pay.order_id = o.id
WHERE o.status = 'refunded'
ORDER BY o.id, pay.paid_at;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id  |  amount  | method |  paid_on
-----+----------+--------+------------
  55 |  1190.00 | card   | 2025-11-10
  55 | -1190.00 | card   | 2025-11-26
  97 |    90.00 | paypal | 2026-03-13
  97 |   -90.00 | paypal | 2026-03-26
 106 |   339.00 | paypal | 2026-04-04
 106 |  -339.00 | paypal | 2026-04-17
(6 rows)
```

Κάθε επιστροφή έχει δύο γραμμές. Την αρχική πληρωμή και ένα αρνητικό ποσό με το ίδιο μέγεθος. Το άθροισμά τους είναι μηδέν, κάτι που θα ελέγξουμε με `SUM` στο κεφάλαιο 10.

Άσκηση 7.7 · Κριτικές με χαμηλή βαθμολογία

Εμφάνισε βαθμολογία, όνομα προϊόντος, όνομα πελάτη και τίτλο για τις κριτικές με βαθμολογία 1 ή 2.

ΛΥΣΗ

```
SELECT r.rating, p.name AS product, c.first_name, r.title
FROM reviews AS r
JOIN products AS p ON p.id = r.product_id
JOIN customers AS c ON c.id = r.customer_id
WHERE r.rating <= 2
ORDER BY r.rating, p.name;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 rating |      product       | first_name |    title
--------+--------------------+------------+-------------
      1 | Ασύρματο ποντίκι   | Σοφία      | Χάλασε
      2 | Καφετιέρα espresso | Παναγιώτης | Διαρροή
      2 | Laptop Aero 14     | Σοφία      | Ζεσταίνεται
      2 | Laptop Student 15  | Κατερίνα   | Αργό
(4 rows)
```

Ο `reviews` έχει δύο foreign keys, οπότε ενώνεται με δύο διαφορετικούς πίνακες. Κάθε κριτική βρίσκει ακριβώς ένα προϊόν και έναν πελάτη.

Άσκηση 7.8 · Αποστολή εκτός περιφέρειας

Βρες τις παραγγελίες που στάλθηκαν σε πόλη άλλης περιφέρειας από την περιφέρεια κατοικίας του πελάτη. Εμφάνισε `id` παραγγελίας και τις δύο περιφέρειες.

ΛΥΣΗ

```
SELECT o.id, home.region AS home_region, ship.region AS ship_region
FROM orders AS o
JOIN customers AS c ON c.id = o.customer_id
JOIN cities AS home ON home.id = c.city_id
JOIN cities AS ship ON ship.id = o.shipping_city_id
WHERE home.region <> ship.region
ORDER BY o.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id  |    home_region     |  ship_region
-----+--------------------+---------------
   7 | Κεντρική Μακεδονία | Θεσσαλία
  21 | Αττική             | Ήπειρος
  31 | Κεντρική Μακεδονία | Κρήτη
  38 | Κεντρική Μακεδονία | Θεσσαλία
  41 | Κρήτη              | Πελοπόννησος
  53 | Ήπειρος            | Αττική
  78 | Κεντρική Μακεδονία | Δυτική Ελλάδα
  83 | Κρήτη              | Πελοπόννησος
  87 | Κρήτη              | Αττική
  89 | Κεντρική Μακεδονία | Κρήτη
  95 | Κρήτη              | Δυτική Ελλάδα
  99 | Δυτική Ελλάδα      | Κρήτη
 129 | Αττική             | Ήπειρος
 133 | Δυτική Ελλάδα      | Πελοπόννησος
 …
(17 rows)
```

Χρειάζεσαι τον `cities` δύο φορές με διαφορετικά aliases. Μια αποστολή από τον Βόλο στη Λάρισα αλλάζει πόλη αλλά όχι περιφέρεια, οπότε δεν εμφανίζεται.

ΚΕΦΑΛΑΙΟ 8

OUTER JOIN και anti join

Άσκηση 8.1 · Πόλη ή άγνωστη

Εμφάνισε `id`, όνομα και πόλη όλων των πελατών που εγγράφηκαν το 2026. Όπου η πόλη λείπει γράψε `άγνωστη`.

ΛΥΣΗ

```
SELECT c.id, c.first_name, COALESCE(t.name, 'άγνωστη') AS city
FROM customers AS c
LEFT JOIN cities AS t ON t.id = c.city_id
WHERE c.created_at >= '2026-01-01'
ORDER BY c.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id | first_name |    city
----+------------+-------------
 27 | Πέτρος     | Θεσσαλονίκη
 28 | Δέσποινα   | άγνωστη
 29 | Ηλίας      | Ηράκλειο
 30 | Όλγα       | Αθήνα
(4 rows)
```

Είναι η άσκηση 7.1 με `LEFT JOIN`. Ο πελάτης 28 επιστρέφει στη λίστα και το `COALESCE` αντικαθιστά το `NULL` της πόλης.

Άσκηση 8.2 · Πόλεις χωρίς πελάτες

Βρες τις πόλεις στις οποίες δεν μένει κανένας πελάτης.

ΛΥΣΗ

```
SELECT t.id, t.name
FROM cities AS t
LEFT JOIN customers AS c ON c.city_id = t.id
WHERE c.id IS NULL;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id |  name
----+--------
 11 | Κοζάνη
(1 row)
```

Αυτή τη φορά αριστερά είναι ο `cities`, γιατί θέλεις να κρατήσεις κάθε πόλη. Η σειρά των πινάκων σε ένα `LEFT JOIN` είναι μέρος της ερώτησης.

Άσκηση 8.3 · Παραγγελίες χωρίς γραμμές

Βρες τις παραγγελίες που δεν έχουν καμία γραμμή στον `order_items`. Εμφάνισε `id`, `status` και `customer_id`.

ΛΥΣΗ

```
SELECT o.id, o.status, o.customer_id
FROM orders AS o
LEFT JOIN order_items AS oi ON oi.order_id = o.id
WHERE oi.order_id IS NULL;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id  | status  | customer_id
-----+---------+-------------
 162 | pending |          12
(1 row)
```

Η στήλη `oi.order_id` είναι μέρος του primary key του `order_items`, οπότε σε πραγματική γραμμή δεν είναι ποτέ `NULL`. Είναι ασφαλής για τον έλεγχο του anti join.

Άσκηση 8.4 · Παραγγελίες χωρίς πληρωμή

Βρες τις παραγγελίες που δεν έχουν πληρωμή και δεν είναι ακυρωμένες. Εμφάνισε `id`, `status` και `created_at`.

ΛΥΣΗ

```
SELECT o.id, o.status, o.created_at
FROM orders AS o
LEFT JOIN payments AS pay ON pay.order_id = o.id
WHERE pay.id IS NULL
  AND o.status <> 'cancelled'
ORDER BY o.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id  | status  |       created_at
-----+---------+------------------------
 150 | shipped | 2026-06-15 18:06:00+03
 157 | pending | 2026-06-23 09:56:00+03
 159 | pending | 2026-06-24 19:00:00+03
 162 | pending | 2026-06-28 17:18:00+03
(4 rows)
```

Η συνθήκη `o.status <> 'cancelled'` αφορά τον αριστερό πίνακα, οπότε μπαίνει στο `WHERE`. Η παραγγελία 150 έχει σταλεί με αντικαταβολή και θα πληρωθεί όταν παραδοθεί. Οι pending δεν έχουν πληρωθεί ακόμη.

Άσκηση 8.5 · Πελάτες και κουπόνια

Εμφάνισε κάθε πελάτη της Αθήνας (πόλη 1) με όνομα και τα `id` των παραγγελιών του που χρησιμοποίησαν κουπόνι. Οι πελάτες χωρίς τέτοια παραγγελία πρέπει να εμφανίζονται μία φορά με ∅. Ταξινόμησε κατά πελάτη και παραγγελία.

ΛΥΣΗ

```
SELECT c.id, c.first_name, o.id AS order_id, o.coupon_code
FROM customers AS c
LEFT JOIN orders AS o ON o.customer_id = c.id AND o.coupon_code IS NOT NULL
WHERE c.city_id = 1
ORDER BY c.id, o.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id | first_name | order_id | coupon_code
----+------------+----------+-------------
  1 | Μαρία      |       11 | WELCOME10
  1 | Μαρία      |       17 | WELCOME10
  1 | Μαρία      |       58 | BLACKFRIDAY
  3 | Ελένη      |        ∅ | ∅
  8 | Παναγιώτης |       13 | WELCOME10
  8 | Παναγιώτης |      148 | WELCOME10
 12 | Αλέξανδρος |       30 | WELCOME10
 19 | Αικατερίνη |       66 | WELCOME10
 23 | Στέλιος    |      108 | WELCOME10
 30 | Όλγα       |        ∅ | ∅
(10 rows)
```

Η συνθήκη του κουπονιού είναι στο `ON`. Αν ήταν στο `WHERE`, οι πελάτες χωρίς κουπόνι θα εξαφανίζονταν.

Άσκηση 8.6 · Υπάλληλοι που είναι και πελάτες

Ο πίνακας `employees` έχει στήλη `email`. Εμφάνισε κάθε υπάλληλο του τμήματος Πωλήσεις και, όπου υπάρχει, τον πελάτη με το ίδιο email. Δεν πρέπει να χαθεί κανένας υπάλληλος.

ΛΥΣΗ

```
SELECT e.full_name, e.email, c.id AS customer_id
FROM employees AS e
LEFT JOIN customers AS c ON c.email = e.email
WHERE e.department = 'Πωλήσεις'
ORDER BY e.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
     full_name      |           email            | customer_id
--------------------+----------------------------+-------------
 Σοφία Κωνσταντίνου | sofia.k@shop.example.gr    |           ∅
 Ιωάννα Ρήγα        | ioanna.r@shop.example.gr   |           ∅
 Στέλιος Παππάς     | stelios.p@shop.example.gr  |           ∅
 Βασιλική Ντόκου    | ∅                          |           ∅
(4 rows)
```

Κανένας υπάλληλος δεν είναι πελάτης, οπότε όλες οι γραμμές έχουν ∅ στο `customer_id`. Ο υπάλληλος με θέση Sales Intern δεν έχει email. Η σύγκριση `NULL = NULL` είναι άγνωστη, οπότε δεν θα ταίριαζε ούτε με πελάτη που επίσης δεν έχει email.

Άσκηση 8.7 · Κατηγορίες και προϊόντα

Εμφάνισε όλες τις κατηγορίες των ηλεκτρονικών (τα `id` από 6 έως 10) με το όνομα κάθε ενεργού προϊόντος τους. Οι κατηγορίες χωρίς ενεργό προϊόν πρέπει να εμφανίζονται με ∅. Ταξινόμησε κατά `id` κατηγορίας και όνομα προϊόντος.

ΛΥΣΗ

```
SELECT cat.id, cat.name AS category, p.name AS product
FROM categories AS cat
LEFT JOIN products AS p ON p.category_id = cat.id AND p.active
WHERE cat.id BETWEEN 6 AND 10
ORDER BY cat.id, p.name;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id |   category    |        product
----+---------------+-----------------------
  6 | Ηλεκτρονικά   | ∅
  7 | Υπολογιστές   | ∅
  8 | Laptops       | Laptop Aero 14
  8 | Laptops       | Laptop Pro 16
  8 | Laptops       | Laptop Student 15
  9 | Περιφερειακά  | Ασύρματο ποντίκι
  9 | Περιφερειακά  | Μηχανικό πληκτρολόγιο
  9 | Περιφερειακά  | Οθόνη 27" 4K
  9 | Περιφερειακά  | USB-C docking station
  9 | Περιφερειακά  | Webcam HD
 10 | Ήχος          | Ακουστικά ANC
 10 | Ήχος          | Ηχείο Bluetooth
 10 | Ήχος          | Μικρόφωνο USB
 10 | Ήχος          | Πικάπ
(14 rows)
```

Το ανενεργό `Laptop Classic 13` φεύγει επειδή το `p.active` είναι στο `ON`. Αν ήταν στο `WHERE`, οι κατηγορίες 6 και 7, που δεν έχουν δικά τους προϊόντα, θα εξαφανίζονταν μαζί του.

Άσκηση 8.8 · Πελάτες χωρίς κριτική

Βρες τους πελάτες που έχουν τουλάχιστον μία παραγγελία `delivered` αλλά δεν έχουν γράψει καμία κριτική. Εμφάνισε `id` και όνομα χωρίς επαναλήψεις.

ΛΥΣΗ

```
SELECT DISTINCT c.id, c.first_name
FROM customers AS c
JOIN orders AS o ON o.customer_id = c.id AND o.status = 'delivered'
LEFT JOIN reviews AS r ON r.customer_id = c.id
WHERE r.id IS NULL
ORDER BY c.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id | first_name
----+------------
 11 | Ιωάννα
 14 | Βασίλης
 15 | Ευαγγελία
 17 | Φωτεινή
 18 | Σπύρος
 19 | Αικατερίνη
 20 | Μιχάλης
 21 | Giorgos
 22 | Άννα
 23 | Στέλιος
 24 | Ζωή
 25 | Λευτέρης
 27 | Πέτρος
 28 | Δέσποινα
 …
(15 rows)
```

Το `INNER JOIN` με τις παραδομένες παραγγελίες κρατά τους πελάτες με τουλάχιστον μία, αλλά βγάζει μία γραμμή για καθεμία. Το `DISTINCT` τις μαζεύει. Εδώ είναι θεμιτό, γιατί θέλουμε πράγματι μια λίστα πελατών. Το κεφάλαιο 13 λύνει το ίδιο χωρίς `DISTINCT`, με `EXISTS`.

ΚΕΦΑΛΑΙΟ 9

Self join, CROSS JOIN και joins με εύρος

Άσκηση 9.1 · Υπάλληλοι και προϊστάμενοι

Εμφάνισε για κάθε υπάλληλο του τμήματος Πωλήσεις το όνομά του, τη θέση του και το όνομα και τη θέση του προϊσταμένου του.

ΛΥΣΗ

```
SELECT e.full_name AS employee, e.title,
       m.full_name AS manager, m.title AS manager_title
FROM employees AS e
LEFT JOIN employees AS m ON m.id = e.manager_id
WHERE e.department = 'Πωλήσεις'
ORDER BY e.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
      employee      |      title      |      manager       |  manager_title
--------------------+-----------------+--------------------+-----------------
 Σοφία Κωνσταντίνου | Head of Sales   | Ελένη Μαυρίδου     | CEO
 Ιωάννα Ρήγα       | Account Manager | Σοφία Κωνσταντίνου | Head of Sales
 Στέλιος Παππάς    | Account Manager | Σοφία Κωνσταντίνου | Head of Sales
 Βασιλική Ντόκου   | Sales Intern    | Ιωάννα Ρήγα        | Account Manager
(4 rows)
```

Εδώ όλοι οι υπάλληλοι των Πωλήσεων έχουν προϊστάμενο, οπότε και το `JOIN` θα έδινε το ίδιο. Το `LEFT JOIN` όμως απαντά σωστά την ερώτηση «για κάθε υπάλληλο» ακόμη κι αν αύριο προστεθεί υπάλληλος χωρίς προϊστάμενο.

Άσκηση 9.2 · Διαφορά μισθού

Για κάθε υπάλληλο με προϊστάμενο και γνωστό μισθό, εμφάνισε το όνομά του, τον μισθό του, τον μισθό του προϊσταμένου και τη διαφορά τους με όνομα `gap`. Ταξινόμησε από τη μεγαλύτερη διαφορά.

ΛΥΣΗ

```
SELECT e.full_name, e.salary, m.salary AS manager_salary,
       m.salary - e.salary AS gap
FROM employees AS e
JOIN employees AS m ON m.id = e.manager_id
WHERE e.salary IS NOT NULL
ORDER BY gap DESC;
```

ΑΠΟΤΕΛΕΣΜΑ

```
       full_name       | salary  | manager_salary |   gap
-----------------------+---------+----------------+---------
 Θανάσης Μπέης         | 1750.00 |        6200.00 | 4450.00
 Γιάννης Κορρές        | 1800.00 |        6200.00 | 4400.00
 Άγγελος Φωτίου        | 3900.00 |        8200.00 | 4300.00
 Στέλιος Παππάς        | 2900.00 |        7000.00 | 4100.00
 Κατερίνα Ζαφειρίου    | 4200.00 |        8200.00 | 4000.00
 Ιωάννα Ρήγα           | 3100.00 |        7000.00 | 3900.00
 Μαρίνα Σταθοπούλου    | 4400.00 |        8200.00 | 3800.00
 Δημήτρης Λαμπρόπουλος | 6200.00 |        9500.00 | 3300.00
 Σοφία Κωνσταντίνου    | 7000.00 |        9500.00 | 2500.00
 Πέτρος Χατζής         | 2100.00 |        4200.00 | 2100.00
 Βασιλική Ντόκου       | 1100.00 |        3100.00 | 2000.00
 Νίκος Αλεξίου         | 8200.00 |        9500.00 | 1300.00
 Χριστίνα Αθανασίου    | 1650.00 |        1800.00 |  150.00
(13 rows)
```

Κανένας υπάλληλος δεν αμείβεται περισσότερο από τον προϊστάμενό του, οπότε όλες οι διαφορές είναι θετικές.

Άσκηση 9.3 · Ζεύγη προϊόντων

Βρες ζεύγη διαφορετικών προϊόντων της ίδιας κατηγορίας που διαφέρουν στην τιμή λιγότερο από 5 ευρώ. Κάθε ζεύγος πρέπει να εμφανίζεται μία φορά. Εμφάνισε τα δύο ονόματα και τις δύο τιμές.

ΛΥΣΗ

```
SELECT a.name, a.price, b.name, b.price
FROM products AS a
JOIN products AS b ON b.category_id = a.category_id
                  AND b.id > a.id
                  AND abs(a.price - b.price) < 5
ORDER BY a.name;
```

ΑΠΟΤΕΛΕΣΜΑ

```
               name                | price |       name        | price
-----------------------------------+-------+-------------------+-------
 Βίος και πολιτεία του Αλέξη Ζορμπά | 16.90 | Ματωμένα χώματα   | 15.20
 Βίος και πολιτεία του Αλέξη Ζορμπά | 16.90 | Το Κιβώτιο        | 14.50
 Μαθαίνω Python                    | 32.00 | Ruby από την αρχή | 29.50
 Το Κιβώτιο                        | 14.50 | Ματωμένα χώματα   | 15.20
 Το Κιβώτιο                        | 14.50 | Η Φόνισσα         |  9.90
(5 rows)
```

Η συνθήκη `b.id > a.id` κρατά κάθε ζεύγος μία φορά. Το `abs` δίνει την απόλυτη τιμή, ώστε να μη μετράει ποιο προϊόν είναι ακριβότερο.

Άσκηση 9.4 · Πλέγμα καταστάσεων και μεθόδων

Φτιάξε με `CROSS JOIN` και δύο λίστες `VALUES` όλους τους συνδυασμούς των καταστάσεων `paid`, `shipped` και `delivered` με τις μεθόδους πληρωμής `card` και `cod`. Ταξινόμησε κατά κατάσταση και μέθοδο.

ΛΥΣΗ

```
SELECT s.status, m.method
FROM (VALUES ('paid'), ('shipped'), ('delivered')) AS s(status)
CROSS JOIN (VALUES ('card'), ('cod')) AS m(method)
ORDER BY s.status, m.method;
```

ΑΠΟΤΕΛΕΣΜΑ

```
  status   | method
-----------+--------
 delivered | card
 delivered | cod
 paid      | card
 paid      | cod
 shipped   | card
 shipped   | cod
(6 rows)
```

Τρεις επί δύο γραμμές δίνουν έξι. Τέτοια πλέγματα γίνονται χρήσιμα όταν ενωθούν με `LEFT JOIN` με πραγματικά δεδομένα, ώστε να φαίνονται και οι συνδυασμοί που δεν συνέβησαν ποτέ.

Άσκηση 9.5 · Ηλικιακές ομάδες

Κατάταξε τους πελάτες που έχουν ημερομηνία γέννησης και `id` έως 10 σε ομάδες `έως 30`, `31 έως 45` και `46 και πάνω`, με βάση την ηλικία τους στις 30 Ιουνίου 2026. Χρησιμοποίησε join με λίστα `VALUES` και όχι `CASE`. Εμφάνισε όνομα, ηλικία και ομάδα.

ΛΥΣΗ

```
SELECT c.first_name,
       extract(year FROM age(date '2026-06-30', c.birth_date)) AS years,
       g.label
FROM customers AS c
JOIN (VALUES ('έως 30', 0, 31),
             ('31 έως 45', 31, 46),
             ('46 και πάνω', 46, 200)) AS g(label, lo, hi)
  ON extract(year FROM age(date '2026-06-30', c.birth_date)) >= g.lo
 AND extract(year FROM age(date '2026-06-30', c.birth_date)) < g.hi
WHERE c.id <= 10
ORDER BY c.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 first_name | years |    label
------------+-------+-------------
 Μαρία      |    38 | 31 έως 45
 Γιώργος    |    46 | 46 και πάνω
 Ελένη      |    31 | 31 έως 45
 Αναστασία  |    24 | έως 30
 Νίκος      |    36 | 31 έως 45
 Δήμητρα    |    40 | 31 έως 45
 Παναγιώτης |    54 | 46 και πάνω
 Σοφία      |    26 | έως 30
 Χρήστος    |    42 | 31 έως 45
(9 rows)
```

Η έκφραση της ηλικίας επαναλαμβάνεται τρεις φορές. Το κεφάλαιο 14 δείχνει πώς την υπολογίζεις μία φορά σε derived table. Ο πελάτης 4 δεν έχει ημερομηνία γέννησης και το join τον αφήνει έξω.

Άσκηση 9.6 · Τρία επίπεδα ιεραρχίας

Εμφάνισε κάθε υπάλληλο που έχει προϊστάμενο και προϊστάμενο του προϊσταμένου, μαζί με τα δύο αυτά ονόματα. Ταξινόμησε κατά `id` υπαλλήλου.

ΛΥΣΗ

```
SELECT e.full_name AS employee, m.full_name AS manager, mm.full_name AS top
FROM employees AS e
JOIN employees AS m ON m.id = e.manager_id
JOIN employees AS mm ON mm.id = m.manager_id
ORDER BY e.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
      employee      |        manager        |          top
--------------------+-----------------------+-----------------------
 Κατερίνα Ζαφειρίου | Νίκος Αλεξίου         | Ελένη Μαυρίδου
 Άγγελος Φωτίου     | Νίκος Αλεξίου         | Ελένη Μαυρίδου
 Μαρίνα Σταθοπούλου | Νίκος Αλεξίου         | Ελένη Μαυρίδου
 Πέτρος Χατζής      | Κατερίνα Ζαφειρίου    | Νίκος Αλεξίου
 Ιωάννα Ρήγα        | Σοφία Κωνσταντίνου    | Ελένη Μαυρίδου
 Στέλιος Παππάς     | Σοφία Κωνσταντίνου    | Ελένη Μαυρίδου
 Βασιλική Ντόκου    | Ιωάννα Ρήγα           | Σοφία Κωνσταντίνου
 Γιάννης Κορρές     | Δημήτρης Λαμπρόπουλος | Ελένη Μαυρίδου
 Θανάσης Μπέης      | Δημήτρης Λαμπρόπουλος | Ελένη Μαυρίδου
 Χριστίνα Αθανασίου | Γιάννης Κορρές        | Δημήτρης Λαμπρόπουλος
(10 rows)
```

Τα δύο `INNER JOIN` κρατούν μόνο όσους έχουν δύο επίπεδα από πάνω τους. Οι άμεσοι υφιστάμενοι του υπαλλήλου 1 δεν εμφανίζονται, γιατί ο προϊστάμενός τους δεν έχει προϊστάμενο.

Άσκηση 9.7 · Πληρωμή μετά την αποστολή

Βρες τις πληρωμές που έγιναν μετά την αποστολή της παραγγελίας τους και είναι θετικές. Εμφάνισε `id` παραγγελίας, μέθοδο, ημερομηνία αποστολής και ημερομηνία πληρωμής ως `date`. Κράτα όσες έγιναν από τον Απρίλιο του 2026.

ΛΥΣΗ

```
SELECT o.id, pay.method, o.shipped_at::date AS shipped, pay.paid_at::date AS paid
FROM orders AS o
JOIN payments AS pay ON pay.order_id = o.id
                    AND pay.paid_at > o.shipped_at
                    AND pay.amount > 0
WHERE pay.paid_at >= '2026-04-01'
ORDER BY pay.paid_at;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id  | method |  shipped   |    paid
-----+--------+------------+------------
 123 | cod    | 2026-05-20 | 2026-05-21
 125 | cod    | 2026-05-25 | 2026-05-26
 128 | cod    | 2026-05-26 | 2026-05-27
 137 | cod    | 2026-06-07 | 2026-06-09
 136 | cod    | 2026-06-09 | 2026-06-10
 151 | cod    | 2026-06-18 | 2026-06-20
 146 | cod    | 2026-06-19 | 2026-06-20
(7 rows)
```

Όλες είναι αντικαταβολές, `cod`, που πληρώνονται όταν παραδοθεί το δέμα. Η συνθήκη του join συγκρίνει δύο χρονικές στιγμές από διαφορετικούς πίνακες, κάτι που καμία ισότητα δεν μπορεί να εκφράσει.

ΚΕΦΑΛΑΙΟ 10

Aggregates και GROUP BY

Άσκηση 10.1 · Στοιχεία πελατών

Σε ένα query μέτρησε τους πελάτες, πόσοι έχουν τηλέφωνο, πόσοι έχουν εγγραφεί στο newsletter και σε πόσες διαφορετικές πόλεις μένουν.

ΛΥΣΗ

```
SELECT count(*) AS customers,
       count(phone) AS with_phone,
       count(CASE WHEN newsletter THEN 1 END) AS newsletter,
       count(DISTINCT city_id) AS cities
FROM customers;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 customers | with_phone | newsletter | cities
-----------+------------+------------+--------
        30 |         22 |         13 |     10
(1 row)
```

Για το newsletter το `CASE` χωρίς `ELSE` δίνει 1 για όσους έχουν εγγραφεί και `NULL` για τους υπόλοιπους. Το `count` αγνοεί τα `NULL`, οπότε μετρά μόνο τους πρώτους. Το επόμενο κεφάλαιο δείχνει μια πιο καθαρή γραφή, το `FILTER`.

Άσκηση 10.2 · Πληρωμές ανά μέθοδο

Για κάθε μέθοδο πληρωμής εμφάνισε πόσες πληρωμές έγιναν και το καθαρό σύνολο, δηλαδή το άθροισμα όλων των ποσών μαζί με τις επιστροφές. Ταξινόμησε από το μεγαλύτερο σύνολο.

ΛΥΣΗ

```
SELECT method, count(*) AS payments, sum(amount) AS net_total
FROM payments
GROUP BY method
ORDER BY net_total DESC;
```

ΑΠΟΤΕΛΕΣΜΑ

```
  method  | payments | net_total
----------+----------+-----------
 card     |       87 |  36905.50
 paypal   |       28 |   9977.40
 bank     |       15 |   8463.50
 cod      |       17 |   7336.80
 giftcard |        8 |    400.00
(5 rows)
```

Οι επιστροφές είναι αρνητικά ποσά, οπότε το `sum` τις αφαιρεί αυτόματα. Το `count(*)` όμως τις μετράει ως πληρωμές. Το επόμενο κεφάλαιο δείχνει πώς μετράς χωριστά χρεώσεις και επιστροφές.

Άσκηση 10.3 · Τιμές ανά κατηγορία

Για κάθε κατηγορία με ενεργά προϊόντα εμφάνισε το όνομά της, πόσα ενεργά προϊόντα έχει, τη μικρότερη, τη μεγαλύτερη και τη μέση τιμή τους σε δύο δεκαδικά. Ταξινόμησε κατά όνομα κατηγορίας.

ΛΥΣΗ

```
SELECT cat.name AS category,
       count(*) AS products,
       min(p.price) AS min_price,
       max(p.price) AS max_price,
       round(avg(p.price), 2) AS avg_price
FROM products AS p
JOIN categories AS cat ON cat.id = p.category_id
WHERE p.active
GROUP BY cat.name
ORDER BY cat.name;
```

ΑΠΟΤΕΛΕΣΜΑ

```
     category     | products | min_price | max_price | avg_price
------------------+----------+-----------+-----------+-----------
 Βάσεις δεδομένων |        2 |     36.00 |     45.00 |     40.50
 Γραφείο          |        3 |     42.00 |    459.00 |    263.33
 Ήχος             |        4 |     79.00 |    229.00 |    156.50
 Κουζίνα          |        3 |     34.90 |    249.00 |    115.97
 Λογοτεχνία       |        4 |      9.90 |     16.90 |     14.13
 Περιφερειακά     |        5 |     24.90 |    329.00 |    128.18
 Προγραμματισμός  |        3 |     29.50 |     38.00 |     33.17
 Τεχνολογία       |        1 |     41.00 |     41.00 |     41.00
 Laptops          |        3 |    649.00 |   2199.00 |   1332.33
(9 rows)
```

Η ομαδοποίηση γίνεται με το όνομα της κατηγορίας, που είναι μοναδικό. Αν δύο κατηγορίες μπορούσαν να έχουν το ίδιο όνομα, θα ομαδοποιούσες με `cat.id, cat.name`.

Άσκηση 10.4 · Έσοδα ανά μήνα

Εμφάνισε για κάθε μήνα του 2026 τον αριθμό των παραγγελιών που δεν ακυρώθηκαν ούτε επιστράφηκαν και τα έσοδά τους. Ο μήνας να εμφανίζεται ως `YYYY-MM`.

ΛΥΣΗ

```
SELECT to_char(o.created_at, 'YYYY-MM') AS month,
       count(DISTINCT o.id) AS orders,
       sum(oi.quantity * oi.unit_price) AS revenue
FROM orders AS o
JOIN order_items AS oi ON oi.order_id = o.id
WHERE o.status NOT IN ('cancelled', 'refunded')
  AND o.created_at >= '2026-01-01'
GROUP BY to_char(o.created_at, 'YYYY-MM')
ORDER BY month;
```

ΑΠΟΤΕΛΕΣΜΑ

```
  month  | orders | revenue
---------+--------+----------
 2026-01 |      6 |  1860.80
 2026-02 |     11 |  3747.20
 2026-03 |     10 |  1599.20
 2026-04 |      8 |  4026.50
 2026-05 |     18 | 10338.80
 2026-06 |     28 |  9074.90
(6 rows)
```

Μετά το join κάθε παραγγελία εμφανίζεται μία φορά για κάθε γραμμή της. Το `count(*)` θα μετρούσε γραμμές παραγγελιών. Το `count(DISTINCT o.id)` μετρά παραγγελίες. Το άθροισμα είναι σωστό, αφού αθροίζει ακριβώς τις γραμμές.

Άσκηση 10.5 · Ιστορικό πελάτη

Για τους πελάτες 1 έως 6 εμφάνισε `id`, όνομα, πλήθος παραγγελιών και τις ημερομηνίες της πρώτης και της τελευταίας ως `date`.

ΛΥΣΗ

```
SELECT c.id, c.first_name,
       count(*) AS orders,
       min(o.created_at)::date AS first_order,
       max(o.created_at)::date AS last_order
FROM customers AS c
JOIN orders AS o ON o.customer_id = c.id
WHERE c.id BETWEEN 1 AND 6
GROUP BY c.id
ORDER BY c.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id | first_name | orders | first_order | last_order
----+------------+--------+-------------+------------
  1 | Μαρία      |     12 | 2025-03-18  | 2026-05-26
  2 | Γιώργος    |      9 | 2025-02-03  | 2026-05-25
  3 | Ελένη      |      8 | 2025-01-28  | 2026-06-14
  4 | Κώστας     |      8 | 2025-01-17  | 2026-06-09
  5 | Αναστασία  |      7 | 2025-02-06  | 2026-06-06
  6 | Νίκος      |     10 | 2025-01-19  | 2026-06-13
(6 rows)
```

Το `GROUP BY c.id` αρκεί για να εμφανιστεί το `c.first_name`, επειδή το `id` είναι primary key. Το `min` και το `max` δουλεύουν σε κάθε τύπο που ταξινομείται, όχι μόνο σε αριθμούς.

Άσκηση 10.6 · Οι μεγαλύτερες παραγγελίες

Εμφάνισε τις πέντε παραγγελίες με τη μεγαλύτερη αξία, μαζί με το όνομα του πελάτη και το πλήθος των τεμαχίων τους.

ΛΥΣΗ

```
SELECT o.id, c.first_name, c.last_name,
       sum(oi.quantity) AS units,
       sum(oi.quantity * oi.unit_price) AS total
FROM orders AS o
JOIN customers AS c ON c.id = o.customer_id
JOIN order_items AS oi ON oi.order_id = o.id
GROUP BY o.id, c.id
ORDER BY total DESC
LIMIT 5;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id  | first_name |  last_name   | units |  total
-----+------------+--------------+-------+---------
 114 | Στέλιος    | Κουτσούκος   |     3 | 3047.00
  24 | Γιώργος    | Παπαδόπουλος |     5 | 3011.00
  67 | Σοφία      | Αλεξίου      |     6 | 2717.70
  40 | Κατερίνα   | Σταύρου      |     4 | 2527.50
  44 | Παναγιώτης | Βασιλείου    |     2 | 2448.00
(5 rows)
```

Η ομαδοποίηση με τα δύο primary keys επιτρέπει να εμφανιστούν στήλες και των δύο πινάκων. Το `LIMIT` εφαρμόζεται μετά την ταξινόμηση των ομάδων.

Άσκηση 10.7 · Τι περιείχε κάθε παραγγελία

Για κάθε παραγγελία του πελάτη 13 εμφάνισε το `id` και τα ονόματα των προϊόντων της σε μία στήλη, αλφαβητικά, χωρισμένα με κόμμα και κενό.

ΛΥΣΗ

```
SELECT o.id, string_agg(p.name, ', ' ORDER BY p.name) AS products
FROM orders AS o
JOIN order_items AS oi ON oi.order_id = o.id
JOIN products AS p ON p.id = oi.product_id
WHERE o.customer_id = 13
GROUP BY o.id
ORDER BY o.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id |                                products
----+-------------------------------------------------------------------------
 20 | Δίκτυα υπολογιστών, Η Φόνισσα, Σετ μαχαιριών
 35 | Ασύρματο ποντίκι, Βίος και πολιτεία του Αλέξη Ζορμπά, Laptop Student 15
 40 | Δωροκάρτα 50€, Καφετιέρα espresso, Laptop Pro 16, Ruby από την αρχή
 41 | Εργονομική καρέκλα
 82 | Μικρόφωνο USB, PostgreSQL σε βάθος
(5 rows)
```

Το `ORDER BY` μέσα στο `string_agg` αφορά μόνο τη σειρά των ονομάτων μέσα στη στήλη. Το εξωτερικό `ORDER BY` ταξινομεί τις γραμμές του αποτελέσματος.

Άσκηση 10.8 · Βαθμολογίες προϊόντων

Για κάθε προϊόν με κριτικές εμφάνισε όνομα, πλήθος κριτικών, μέση βαθμολογία με ένα δεκαδικό και τη χαμηλότερη βαθμολογία. Κράτα μόνο τα προϊόντα της κατηγορίας 10 και ταξινόμησε από τη μεγαλύτερη μέση βαθμολογία.

ΛΥΣΗ

```
SELECT p.name, count(*) AS reviews,
       round(avg(r.rating), 1) AS avg_rating,
       min(r.rating) AS worst
FROM reviews AS r
JOIN products AS p ON p.id = r.product_id
WHERE p.category_id = 10
GROUP BY p.id
ORDER BY avg_rating DESC;
```

ΑΠΟΤΕΛΕΣΜΑ

```
      name       | reviews | avg_rating | worst
-----------------+---------+------------+-------
 Μικρόφωνο USB   |       1 |        5.0 |     5
 Ακουστικά ANC   |       4 |        4.3 |     3
 Ηχείο Bluetooth |       1 |        4.0 |     4
(3 rows)
```

Το `avg` σε ακέραιο `smallint` επιστρέφει `numeric`, οπότε δεν χάνεις δεκαδικά. Η ακέραια διαίρεση του κεφαλαίου 5 δεν σε αφορά εδώ, αλλά θα σε αφορούσε αν έγραφες `sum(r.rating) / count(*)`.

ΚΕΦΑΛΑΙΟ 11

HAVING, FILTER και η παγίδα του fan out

Άσκηση 11.1 · Οι πιστοί πελάτες

Εμφάνισε `id`, όνομα, επώνυμο και πλήθος παραγγελιών των πελατών με τουλάχιστον εννέα παραγγελίες, από τον πιο δραστήριο.

ΛΥΣΗ

```
SELECT c.id, c.first_name, c.last_name, count(*) AS orders
FROM customers AS c
JOIN orders AS o ON o.customer_id = c.id
GROUP BY c.id
HAVING count(*) >= 9
ORDER BY orders DESC, c.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id | first_name |  last_name   | orders
----+------------+--------------+--------
  1 | Μαρία      | Παπαδοπούλου |     12
  6 | Νίκος      | Ιωάννου      |     10
 12 | Αλέξανδρος | Οικονόμου    |     10
  2 | Γιώργος    | Παπαδόπουλος |      9
(4 rows)
```

Το `HAVING` επαναλαμβάνει το `count(*)` αντί για το alias `orders`. Το `HAVING` εκτελείται πριν από το `SELECT`, οπότε το alias δεν υπάρχει ακόμη.

Άσκηση 11.2 · Κατηγορίες με μεγάλο τζίρο

Εμφάνισε τις κατηγορίες με έσοδα πάνω από 4000 ευρώ από παραγγελίες που δεν ακυρώθηκαν ούτε επιστράφηκαν, με τα έσοδά τους.

ΛΥΣΗ

```
SELECT cat.name AS category, sum(oi.quantity * oi.unit_price) AS revenue
FROM order_items AS oi
JOIN orders AS o ON o.id = oi.order_id
JOIN products AS p ON p.id = oi.product_id
JOIN categories AS cat ON cat.id = p.category_id
WHERE o.status NOT IN ('cancelled', 'refunded')
GROUP BY cat.name
HAVING sum(oi.quantity * oi.unit_price) > 4000
ORDER BY revenue DESC;
```

ΑΠΟΤΕΛΕΣΜΑ

```
   category    | revenue
---------------+----------
 Laptops       | 36572.00
 Γραφείο       |  8800.00
 Περιφερειακά  |  6675.10
 Ήχος          |  4985.00
(4 rows)
```

Εδώ το join είναι αλυσίδα από «πολλά» προς «ένα». Κάθε γραμμή του `order_items` βρίσκει μία παραγγελία, ένα προϊόν και μία κατηγορία. Το grain μένει μία γραμμή ανά γραμμή παραγγελίας και το άθροισμα είναι σωστό.

Άσκηση 11.3 · Χρεώσεις και επιστροφές

Για τους πελάτες που έχουν τουλάχιστον μία επιστροφή, εμφάνισε `customer_id`, το σύνολο των θετικών πληρωμών τους και το σύνολο των επιστροφών.

ΛΥΣΗ

```
SELECT o.customer_id,
       sum(pay.amount) FILTER (WHERE pay.amount > 0) AS charged,
       sum(pay.amount) FILTER (WHERE pay.amount < 0) AS refunded
FROM orders AS o
JOIN payments AS pay ON pay.order_id = o.id
GROUP BY o.customer_id
HAVING count(*) FILTER (WHERE pay.amount < 0) > 0
ORDER BY o.customer_id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 customer_id | charged | refunded
-------------+---------+----------
           4 | 3539.90 |   -90.00
           6 | 2243.50 | -1190.00
          28 | 1021.00 |  -339.00
(3 rows)
```

Και το `HAVING` δέχεται aggregate με `FILTER`. Εδώ υπάρχει ένα μόνο join προς πίνακα «πολλά», οπότε δεν υπάρχει fan out.

Άσκηση 11.4 · Δημοφιλή προϊόντα

Βρες τα προϊόντα που αγόρασαν τουλάχιστον 12 διαφορετικοί πελάτες. Εμφάνισε όνομα, πλήθος πελατών και συνολικά τεμάχια.

ΛΥΣΗ

```
SELECT p.name,
       count(DISTINCT o.customer_id) AS customers,
       sum(oi.quantity) AS units
FROM order_items AS oi
JOIN orders AS o ON o.id = oi.order_id
JOIN products AS p ON p.id = oi.product_id
GROUP BY p.id
HAVING count(DISTINCT o.customer_id) >= 12
ORDER BY p.name;
```

ΑΠΟΤΕΛΕΣΜΑ

```
        name         | customers | units
---------------------+-----------+-------
 Ακουστικά ANC       |        12 |    14
 Μικρόφωνο USB       |        12 |    16
 PostgreSQL σε βάθος |        12 |    15
(3 rows)
```

Ο ίδιος πελάτης μπορεί να αγόρασε το ίδιο προϊόν σε πολλές παραγγελίες. Το `count(*)` θα μετρούσε αγορές. Το `count(DISTINCT o.customer_id)` μετρά ανθρώπους.

Άσκηση 11.5 · Μήνες με κίνηση

Εμφάνισε τους μήνες του 2025 με τουλάχιστον 10 παραγγελίες. Για καθέναν δείξε τον μήνα ως `YYYY-MM`, το σύνολο των παραγγελιών και πόσες από αυτές είχαν κουπόνι.

ΛΥΣΗ

```
SELECT to_char(created_at, 'YYYY-MM') AS month,
       count(*) AS orders,
       count(*) FILTER (WHERE coupon_code IS NOT NULL) AS with_coupon
FROM orders
WHERE created_at >= '2025-01-01' AND created_at < '2026-01-01'
GROUP BY to_char(created_at, 'YYYY-MM')
HAVING count(*) >= 10
ORDER BY month;
```

ΑΠΟΤΕΛΕΣΜΑ

```
  month  | orders | with_coupon
---------+--------+-------------
 2025-09 |     10 |           0
 2025-11 |     12 |           4
 2025-12 |     11 |           2
(3 rows)
```

Ο Νοέμβριος έχει τα περισσότερα κουπόνια λόγω της Black Friday. Το `FILTER` μετρά μόνο αυτά, ενώ το `HAVING` κοιτάζει το σύνολο.

Άσκηση 11.6 · Παραγγελίες και κριτικές

Για τους πελάτες 1 έως 5 εμφάνισε `id`, πλήθος παραγγελιών και πλήθος κριτικών. Πελάτες χωρίς κριτικές πρέπει να έχουν 0. Πρόσεξε το fan out.

ΛΥΣΗ

```
SELECT c.id,
       count(DISTINCT o.id) AS orders,
       count(DISTINCT r.id) AS reviews
FROM customers AS c
LEFT JOIN orders AS o ON o.customer_id = c.id
LEFT JOIN reviews AS r ON r.customer_id = c.id
WHERE c.id BETWEEN 1 AND 5
GROUP BY c.id
ORDER BY c.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id | orders | reviews
----+--------+---------
  1 |     12 |       3
  2 |      9 |       8
  3 |      8 |       4
  4 |      8 |       3
  5 |      7 |       2
(5 rows)
```

Τα δύο `LEFT JOIN` φτιάχνουν κάθε συνδυασμό παραγγελίας και κριτικής του ίδιου πελάτη. Για μετρήσεις το `count(DISTINCT ...)` με primary key διορθώνει το πρόβλημα. Το `count(DISTINCT r.id)` δίνει 0 όταν όλα τα `r.id` είναι `NULL`, ενώ το `count(*)` θα έδινε τουλάχιστον 1.

Άσκηση 11.7 · Απλήρωτα υπόλοιπα

Βρες τις παραγγελίες με κατάσταση `delivered` όπου το ποσό που πληρώθηκε διαφέρει από την αξία των προϊόντων. Άθροισε κάθε πλευρά χωριστά πριν από το join. Αν δεν βρεις καμία, το αποτέλεσμα είναι σωστό.

ΛΥΣΗ

```
SELECT o.id, it.items_total, COALESCE(pt.paid_total, 0) AS paid_total
FROM orders AS o
JOIN (SELECT order_id, sum(quantity * unit_price) AS items_total
      FROM order_items
      GROUP BY order_id) AS it ON it.order_id = o.id
LEFT JOIN (SELECT order_id, sum(amount) AS paid_total
           FROM payments
           GROUP BY order_id) AS pt ON pt.order_id = o.id
WHERE o.status = 'delivered'
  AND it.items_total <> COALESCE(pt.paid_total, 0);
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id | items_total | paid_total
----+-------------+------------
(0 rows)
```

Το αποτέλεσμα είναι άδειο. Κάθε παραδομένη παραγγελία έχει πληρωθεί ακριβώς. Με το naive query με δύο joins θα έβρισκες δεκάδες «διαφορές» που δεν υπάρχουν. Τέτοια queries συμφωνίας είναι από τα πιο χρήσιμα που θα γράψεις, με την προϋπόθεση ότι αθροίζουν σωστά.

ΚΕΦΑΛΑΙΟ 12

UNION, INTERSECT και EXCEPT

Άσκηση 12.1 · Ενιαίος κατάλογος email

Φτιάξε μια λίστα με τα email των υπαλλήλων του τμήματος Τεχνολογία και των προμηθευτών, όπου υπάρχει email, με δεύτερη στήλη `kind` που λέει `υπάλληλος` ή `προμηθευτής`. Ταξινόμησε κατά email.

ΛΥΣΗ

```
SELECT email, 'υπάλληλος' AS kind
FROM employees
WHERE department = 'Τεχνολογία' AND email IS NOT NULL
UNION ALL
SELECT email, 'προμηθευτής'
FROM suppliers
WHERE email IS NOT NULL
ORDER BY email;
```

ΑΠΟΤΕΛΕΣΜΑ

```
              email               |    kind
----------------------------------+-------------
 angelos.f@shop.example.gr        | υπάλληλος
 hello@nordicoffice.example.com   | προμηθευτής
 info@kritiki.example.gr          | προμηθευτής
 katerina.z@shop.example.gr       | υπάλληλος
 marina.s@shop.example.gr         | υπάλληλος
 nikos.a@shop.example.gr          | υπάλληλος
 orders@vivliodianomi.example.gr  | προμηθευτής
 petros.ch@shop.example.gr        | υπάλληλος
 sales@technoimport.example.gr    | προμηθευτής
 xartika@example.gr               | προμηθευτής
(10 rows)
```

Η στήλη `kind` παίρνει όνομα από το πρώτο query. Στο δεύτερο αρκεί η τιμή.

Άσκηση 12.2 · Πόλεις με πελάτες ή προμηθευτές

Εμφάνισε χωρίς επαναλήψεις τα ονόματα των πόλεων όπου μένει τουλάχιστον ένας πελάτης ή έχει έδρα τουλάχιστον ένας προμηθευτής, ταξινομημένα.

ΛΥΣΗ

```
SELECT t.name
FROM customers AS c
JOIN cities AS t ON t.id = c.city_id
UNION
SELECT t.name
FROM suppliers AS s
JOIN cities AS t ON t.id = s.city_id
ORDER BY name;
```

ΑΠΟΤΕΛΕΣΜΑ

```
    name
-------------
 Αθήνα
 Βόλος
 Ηράκλειο
 Θεσσαλονίκη
 Ιωάννινα
 Καλαμάτα
 Λάρισα
 Πάτρα
 Ρόδος
 Χανιά
(10 rows)
```

Εδώ το `UNION` χωρίς `ALL` είναι το σωστό, γιατί θέλουμε κάθε πόλη μία φορά. Κάθε κομμάτι μπορεί να περιέχει joins, `WHERE` και `GROUP BY`.

Άσκηση 12.3 · Προϊόντα του 2025 που δεν ξαναπουλήθηκαν

Βρες τα `product_id` που πουλήθηκαν σε παραγγελίες του 2025 αλλά σε καμία παραγγελία του 2026.

ΛΥΣΗ

```
SELECT oi.product_id
FROM order_items AS oi
JOIN orders AS o ON o.id = oi.order_id
WHERE o.created_at >= '2025-01-01' AND o.created_at < '2026-01-01'
EXCEPT
SELECT oi.product_id
FROM order_items AS oi
JOIN orders AS o ON o.id = oi.order_id
WHERE o.created_at >= '2026-01-01'
ORDER BY product_id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 product_id
------------
         29
(1 row)
```

Το ανενεργό `Laptop Classic 13` (29) πουλήθηκε μόνο στις αρχές του 2025. Η δωροκάρτα (28) δεν αγοράστηκε το 2026.

Άσκηση 12.4 · Μέθοδοι πληρωμής και στα δύο έτη

Βρες τις μεθόδους πληρωμής που χρησιμοποιήθηκαν και τον Δεκέμβριο του 2025 και τον Ιούνιο του 2026.

ΛΥΣΗ

```
SELECT method FROM payments
WHERE paid_at >= '2025-12-01' AND paid_at < '2026-01-01'
INTERSECT
SELECT method FROM payments
WHERE paid_at >= '2026-06-01' AND paid_at < '2026-07-01'
ORDER BY method;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 method
--------
 bank
 card
 cod
 paypal
(4 rows)
```

Το `INTERSECT` αφαιρεί τα διπλότυπα, οπότε κάθε μέθοδος εμφανίζεται μία φορά όσες πληρωμές κι αν έχει.

Άσκηση 12.5 · Πελάτες που χάθηκαν, με όνομα

Εμφάνισε `id`, όνομα και επώνυμο των πελατών που αγόρασαν το δεύτερο εξάμηνο του 2025 αλλά όχι τους πέντε πρώτους μήνες του 2026. Υπολόγισε τη λίστα των `id` με `EXCEPT` και κάνε join με τον `customers`.

ΛΥΣΗ

```
SELECT c.id, c.first_name, c.last_name
FROM customers AS c
JOIN (SELECT customer_id FROM orders
      WHERE created_at >= '2025-07-01' AND created_at < '2026-01-01'
      EXCEPT
      SELECT customer_id FROM orders
      WHERE created_at >= '2026-01-01' AND created_at < '2026-06-01') AS lost
  ON lost.customer_id = c.id
ORDER BY c.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id | first_name |  last_name
----+------------+-------------
 17 | Φωτεινή    | Λαμπράκη
 18 | Σπύρος     | Μαρκόπουλος
(2 rows)
```

Η πράξη συνόλων μπαίνει σε παρενθέσεις στο `JOIN`, όπως τα εσωτερικά queries του κεφαλαίου 11. Το αποτέλεσμά της συμπεριφέρεται σαν πίνακας με μία στήλη.

Άσκηση 12.6 · Αναφορά με γραμμή συνόλου

Εμφάνισε το πλήθος των παραγγελιών ανά κατάσταση και στο τέλος μια γραμμή `σύνολο` με όλες τις παραγγελίες. Οι καταστάσεις να είναι ταξινομημένες αλφαβητικά πάνω από το σύνολο.

ΛΥΣΗ

```
SELECT status, count(*) AS orders, 1 AS sort_key
FROM orders
GROUP BY status
UNION ALL
SELECT 'σύνολο', count(*), 2
FROM orders
ORDER BY sort_key, status;
```

ΑΠΟΤΕΛΕΣΜΑ

```
  status   | orders | sort_key
-----------+--------+----------
 cancelled |     14 |        1
 delivered |    125 |        1
 paid      |      5 |        1
 pending   |      3 |        1
 refunded  |      3 |        1
 shipped   |     12 |        1
 σύνολο    |    162 |        2
(7 rows)
```

Χωρίς τη `sort_key` η λέξη `σύνολο` θα ταξινομούνταν ανάμεσα στις λατινικές λέξεις με βάση το collation. Μια βοηθητική στήλη ταξινόμησης είναι ο πιο αξιόπιστος τρόπος να ελέγξεις τη θέση μιας ειδικής γραμμής.

ΚΕΦΑΛΑΙΟ 13

Subqueries με IN και EXISTS

Άσκηση 13.1 · Πάνω από τον μέσο μισθό

Εμφάνισε όνομα και μισθό των υπαλλήλων που αμείβονται περισσότερο από τον μέσο μισθό όλων των υπαλλήλων με γνωστό μισθό.

ΛΥΣΗ

```
SELECT full_name, salary
FROM employees
WHERE salary > (SELECT avg(salary) FROM employees)
ORDER BY salary DESC;
```

ΑΠΟΤΕΛΕΣΜΑ

```
       full_name       | salary
-----------------------+---------
 Ελένη Μαυρίδου        | 9500.00
 Νίκος Αλεξίου         | 8200.00
 Σοφία Κωνσταντίνου    | 7000.00
 Δημήτρης Λαμπρόπουλος | 6200.00
 Μαρίνα Σταθοπούλου    | 4400.00
 Κατερίνα Ζαφειρίου    | 4200.00
(6 rows)
```

Το `avg` αγνοεί το `NULL` του συμβούλου, οπότε το subquery είναι ήδη ο μέσος όρος όσων έχουν μισθό. Ο σύμβουλος δεν εμφανίζεται, γιατί το `NULL > x` είναι άγνωστο.

Άσκηση 13.2 · Όσοι έκριναν τα ακουστικά

Εμφάνισε `id` και όνομα των πελατών που έγραψαν κριτική για το προϊόν με όνομα `Ακουστικά ANC`. Χρησιμοποίησε `IN` και όχι το `id` του προϊόντος.

ΛΥΣΗ

```
SELECT id, first_name
FROM customers
WHERE id IN (SELECT r.customer_id
             FROM reviews AS r
             JOIN products AS p ON p.id = r.product_id
             WHERE p.name = 'Ακουστικά ANC')
ORDER BY id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id | first_name
----+------------
  1 | Μαρία
  6 | Νίκος
  7 | Δήμητρα
  8 | Παναγιώτης
(4 rows)
```

Αν ένας πελάτης είχε γράψει δύο κριτικές για τα ακουστικά, το `IN` θα τον επέστρεφε μία φορά. Το join με τον `customers` θα τον έδειχνε δύο φορές.

Άσκηση 13.3 · Πελάτες με επιστροφή

Με `EXISTS`, βρες τους πελάτες που έχουν τουλάχιστον μία παραγγελία `refunded`. Εμφάνισε `id`, όνομα και επώνυμο.

ΛΥΣΗ

```
SELECT c.id, c.first_name, c.last_name
FROM customers AS c
WHERE EXISTS (SELECT 1
              FROM orders AS o
              WHERE o.customer_id = c.id AND o.status = 'refunded')
ORDER BY c.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id | first_name | last_name
----+------------+-----------
  4 | Κώστας     | Δημητρίου
  6 | Νίκος      | Ιωάννου
 28 | Δέσποινα   | Μαυρίδου
(3 rows)
```

Η συνθήκη `o.customer_id = c.id` συνδέει το subquery με την εξωτερική γραμμή. Αν την ξεχνούσες, το `EXISTS` θα ήταν αληθές για όλους, αφού υπάρχουν επιστροφές στον πίνακα.

Άσκηση 13.4 · Πελάτες χωρίς κριτική

Με `NOT EXISTS`, βρες τους πελάτες με `id` έως 12 που δεν έχουν γράψει καμία κριτική.

ΛΥΣΗ

```
SELECT c.id, c.first_name
FROM customers AS c
WHERE c.id <= 12
  AND NOT EXISTS (SELECT 1 FROM reviews AS r WHERE r.customer_id = c.id)
ORDER BY c.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id | first_name
----+------------
 10 | Χρήστος
 11 | Ιωάννα
(2 rows)
```

Είναι η άσκηση 8.8 χωρίς `DISTINCT` και χωρίς join με τις παραγγελίες. Οι πελάτες 10 και 11 δεν έχουν γράψει κριτική.

Άσκηση 13.5 · Πόλεις χωρίς προμηθευτή, σωστά

Εμφάνισε `id` και όνομα των πόλεων στις οποίες δεν έχει έδρα κανένας προμηθευτής αλλά μένει τουλάχιστον ένας πελάτης.

ΛΥΣΗ

```
SELECT t.id, t.name
FROM cities AS t
WHERE NOT EXISTS (SELECT 1 FROM suppliers AS s WHERE s.city_id = t.id)
  AND EXISTS (SELECT 1 FROM customers AS c WHERE c.city_id = t.id)
ORDER BY t.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id |   name
----+----------
  3 | Πάτρα
  6 | Βόλος
  7 | Ιωάννινα
  8 | Χανιά
  9 | Καλαμάτα
 10 | Ρόδος
(6 rows)
```

Δύο subqueries συνδέονται με `AND` όπως κάθε άλλη συνθήκη. Η Κοζάνη δεν έχει ούτε προμηθευτή ούτε πελάτη, οπότε το `EXISTS` την αποκλείει.

Άσκηση 13.6 · Οι παραγγελίες της τελευταίας ημέρας

Εμφάνισε `id`, `customer_id` και `created_at` των παραγγελιών που έγιναν την ίδια ημερολογιακή ημέρα με την πιο πρόσφατη παραγγελία.

ΛΥΣΗ

```
SELECT id, customer_id, created_at
FROM orders
WHERE created_at >= (SELECT date_trunc('day', max(created_at)) FROM orders)
ORDER BY created_at;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id  | customer_id |       created_at
-----+-------------+------------------------
 162 |          12 | 2026-06-28 17:18:00+03
(1 row)
```

Το subquery βρίσκει τη στιγμή της πιο πρόσφατης παραγγελίας και την κόβει στα μεσάνυχτα εκείνης της ημέρας. Κάθε παραγγελία από εκείνη τη στιγμή και μετά ανήκει στην τελευταία ημέρα.

Άσκηση 13.7 · Ακριβότερα από κάθε είδος κουζίνας

Με `ALL`, βρες τα ενεργά προϊόντα που κοστίζουν περισσότερο από κάθε προϊόν της κατηγορίας Κουζίνα (12). Εμφάνισε όνομα και τιμή.

ΛΥΣΗ

```
SELECT name, price
FROM products
WHERE active
  AND price > ALL (SELECT price FROM products WHERE category_id = 12)
ORDER BY price;
```

ΑΠΟΤΕΛΕΣΜΑ

```
            name             |  price
-----------------------------+---------
 Εργονομική καρέκλα          |  289.00
 Οθόνη 27" 4K                |  329.00
 Γραφείο ρυθμιζόμενου ύψους |  459.00
 Laptop Student 15           |  649.00
 Laptop Aero 14              | 1149.00
 Laptop Pro 16               | 2199.00
(6 rows)
```

Η ακριβότερη καφετιέρα κοστίζει 249 ευρώ, οπότε μένουν τα προϊόντα από 249 και πάνω, χωρίς το ίδιο το 249. Το `price > (SELECT max(price) ...)` δίνει το ίδιο αποτέλεσμα.

Άσκηση 13.8 · Παραγγελίες πάνω από τον μέσο όρο

Εμφάνισε `id` και αξία των παραγγελιών του 2026 των οποίων η αξία ξεπερνά τη μέση αξία όλων των παραγγελιών του καταστήματος κατά περισσότερο από 1000 ευρώ.

ΛΥΣΗ

```
SELECT oi.order_id, sum(oi.quantity * oi.unit_price) AS total
FROM order_items AS oi
JOIN orders AS o ON o.id = oi.order_id
WHERE o.created_at >= '2026-01-01'
GROUP BY oi.order_id
HAVING sum(oi.quantity * oi.unit_price) >
       (SELECT avg(t.total) + 1000
        FROM (SELECT order_id, sum(quantity * unit_price) AS total
              FROM order_items
              GROUP BY order_id) AS t)
ORDER BY total DESC;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 order_id |  total
----------+---------
      114 | 3047.00
      130 | 2297.90
      110 | 2208.90
      141 | 2199.00
(4 rows)
```

Το scalar subquery μπαίνει στο `HAVING` όπως θα έμπαινε μια σταθερά. Μέσα του υπάρχει ένα δεύτερο subquery, το ίδιο με το τελευταίο παράδειγμα της θεωρίας.

ΚΕΦΑΛΑΙΟ 14

Correlated subqueries, derived tables και LATERAL

Άσκηση 14.1 · Προϊόντα ανά κατηγορία

Για κάθε κατηγορία εμφάνισε το όνομά της και με correlated subquery το πλήθος των ενεργών προϊόντων της. Οι κατηγορίες χωρίς προϊόντα πρέπει να έχουν 0. Ταξινόμησε κατά `id`.

ΛΥΣΗ

```
SELECT cat.id, cat.name,
       (SELECT count(*)
        FROM products AS p
        WHERE p.category_id = cat.id AND p.active) AS products
FROM categories AS cat
ORDER BY cat.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id |       name        | products
----+-------------------+----------
  1 | Βιβλία            |        0
  2 | Λογοτεχνία        |        4
  3 | Τεχνολογία        |        1
  4 | Προγραμματισμός   |        3
  5 | Βάσεις δεδομένων  |        2
  6 | Ηλεκτρονικά       |        0
  7 | Υπολογιστές       |        0
  8 | Laptops           |        3
  9 | Περιφερειακά      |        5
 10 | Ήχος              |        4
 11 | Σπίτι             |        0
 12 | Κουζίνα           |        3
 13 | Γραφείο           |        3
(13 rows)
```

Το `count(*)` σε subquery που δεν βρίσκει γραμμές δίνει 0. Με `LEFT JOIN` και `count(*)` θα έπαιρνες 1 για τις άδειες κατηγορίες, εκτός αν έγραφες `count(p.id)`.

Άσκηση 14.2 · Φθηνότερα από τον μέσο όρο της κατηγορίας

Εμφάνισε όνομα, κατηγορία και τιμή των ενεργών προϊόντων που κοστίζουν λιγότερο από τον μέσο όρο των ενεργών προϊόντων της κατηγορίας τους.

ΛΥΣΗ

```
SELECT p.name, p.category_id, p.price
FROM products AS p
WHERE p.active
  AND p.price < (SELECT avg(p2.price)
                 FROM products AS p2
                 WHERE p2.category_id = p.category_id AND p2.active)
ORDER BY p.category_id, p.price;
```

ΑΠΟΤΕΛΕΣΜΑ

```
           name             | category_id |  price
----------------------------+-------------+---------
 Η Φόνισσα                  |           2 |    9.90
 Ruby από την αρχή          |           4 |   29.50
 Μαθαίνω Python             |           4 |   32.00
 Σχεδίαση βάσεων δεδομένων  |           5 |   36.00
 Laptop Student 15          |           8 |  649.00
 Laptop Aero 14             |           8 | 1149.00
 Ασύρματο ποντίκι           |           9 |   24.90
 Webcam HD                  |           9 |   59.00
 Μηχανικό πληκτρολόγιο      |           9 |   89.00
 Ηχείο Bluetooth            |          10 |   79.00
 Μικρόφωνο USB              |          10 |  119.00
 Βραστήρας                  |          12 |   34.90
 Σετ μαχαιριών              |          12 |   64.00
 Φωτιστικό γραφείου LED     |          13 |   42.00
(14 rows)
```

Το `p2.active` μέσα στο subquery είναι απαραίτητο για να συγκρίνεις με τον σωστό μέσο όρο. Χωρίς αυτό το ανενεργό `Laptop Classic 13` θα κατέβαζε τον μέσο όρο των laptops.

Άσκηση 14.3 · Κάρτα πελάτη

Για τους πελάτες 25 έως 30 εμφάνισε `id`, όνομα, το συνολικό καθαρό ποσό που έχουν πληρώσει και την ημερομηνία της τελευταίας πληρωμής. Όσοι δεν έχουν πληρώσει πρέπει να έχουν 0 και ∅.

ΛΥΣΗ

```
SELECT c.id, c.first_name,
       (SELECT COALESCE(sum(pay.amount), 0)
        FROM payments AS pay
        JOIN orders AS o ON o.id = pay.order_id
        WHERE o.customer_id = c.id) AS paid,
       (SELECT max(pay.paid_at)::date
        FROM payments AS pay
        JOIN orders AS o ON o.id = pay.order_id
        WHERE o.customer_id = c.id) AS last_payment
FROM customers AS c
WHERE c.id BETWEEN 25 AND 30
ORDER BY c.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id | first_name |  paid   | last_payment
----+------------+---------+--------------
 25 | Λευτέρης   | 1424.90 | 2026-06-09
 26 | Ραλλού     |       0 | ∅
 27 | Πέτρος     |  557.80 | 2026-06-25
 28 | Δέσποινα   |  682.00 | 2026-06-17
 29 | Ηλίας      |  850.40 | 2026-06-19
 30 | Όλγα       |       0 | ∅
(6 rows)
```

Το `COALESCE` είναι μέσα στο subquery, γύρω από το `sum`. Ένα subquery μπορεί να περιέχει joins, όπως εδώ με τον `orders`.

Άσκηση 14.4 · Ηλικιακές ομάδες, χωρίς επανάληψη

Ξαναγράψε την άσκηση 9.5 ώστε η ηλικία να υπολογίζεται μία φορά σε derived table. Εμφάνισε πλήθος πελατών ανά ομάδα για όλους τους πελάτες με γνωστή ημερομηνία γέννησης, ταξινομημένα κατά το κάτω όριο της ομάδας.

ΛΥΣΗ

```
SELECT g.label, count(*) AS customers
FROM (SELECT extract(year FROM age(date '2026-06-30', birth_date)) AS years
      FROM customers
      WHERE birth_date IS NOT NULL) AS a
JOIN (VALUES ('έως 30', 0, 31),
             ('31 έως 45', 31, 46),
             ('46 και πάνω', 46, 200)) AS g(label, lo, hi)
  ON a.years >= g.lo AND a.years < g.hi
GROUP BY g.label, g.lo
ORDER BY g.lo;
```

ΑΠΟΤΕΛΕΣΜΑ

```
    label    | customers
-------------+-----------
 έως 30      |         6
 31 έως 45   |        13
 46 και πάνω |         7
(3 rows)
```

Το `g.lo` μπαίνει στο `GROUP BY` για να μπορεί να χρησιμοποιηθεί στο `ORDER BY`. Κάθε ετικέτα έχει ένα μόνο `lo`, οπότε οι ομάδες δεν αλλάζουν.

Άσκηση 14.5 · Τα δύο ακριβότερα ανά κατηγορία

Για τις κατηγορίες 8, 9 και 10 εμφάνισε το όνομα της κατηγορίας και τα δύο ακριβότερα ενεργά προϊόντα της με την τιμή τους.

ΛΥΣΗ

```
SELECT cat.name AS category, top2.name, top2.price
FROM categories AS cat
CROSS JOIN LATERAL (SELECT p.name, p.price
                    FROM products AS p
                    WHERE p.category_id = cat.id AND p.active
                    ORDER BY p.price DESC
                    LIMIT 2) AS top2
WHERE cat.id IN (8, 9, 10)
ORDER BY cat.id, top2.price DESC;
```

ΑΠΟΤΕΛΕΣΜΑ

```
   category   |         name          |  price
--------------+-----------------------+---------
 Laptops      | Laptop Pro 16         | 2199.00
 Laptops      | Laptop Aero 14        | 1149.00
 Περιφερειακά | Οθόνη 27" 4K          |  329.00
 Περιφερειακά | USB-C docking station |  139.00
 Ήχος         | Πικάπ                 |  229.00
 Ήχος         | Ακουστικά ANC         |  199.00
(6 rows)
```

Το `LIMIT 2` εφαρμόζεται χωριστά για κάθε κατηγορία, επειδή το subquery εκτελείται λογικά μία φορά ανά γραμμή του `cat`.

Άσκηση 14.6 · Ο νεότερος πελάτης κάθε πόλης

Για κάθε πόλη εμφάνισε το όνομά της και τον πελάτη που εγγράφηκε πιο πρόσφατα με την ημερομηνία εγγραφής. Οι πόλεις χωρίς πελάτες πρέπει να εμφανίζονται με ∅. Ταξινόμησε κατά `id` πόλης.

ΛΥΣΗ

```
SELECT t.name AS city, newest.first_name, newest.created_at::date AS joined
FROM cities AS t
LEFT JOIN LATERAL (SELECT c.first_name, c.created_at
                   FROM customers AS c
                   WHERE c.city_id = t.id
                   ORDER BY c.created_at DESC
                   LIMIT 1) AS newest ON true
ORDER BY t.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
    city     | first_name |   joined
-------------+------------+------------
 Αθήνα       | Όλγα       | 2026-06-20
 Θεσσαλονίκη | Πέτρος     | 2026-01-15
 Πάτρα       | Θανάσης    | 2025-06-12
 Ηράκλειο    | Ηλίας      | 2026-04-03
 Λάρισα      | Μιχάλης    | 2025-08-28
 Βόλος       | Άννα       | 2025-09-26
 Ιωάννινα    | Ραλλού     | 2025-12-19
 Χανιά       | Λευτέρης   | 2025-11-27
 Καλαμάτα    | Ευαγγελία  | 2025-05-30
 Ρόδος       | Σπύρος     | 2025-07-19
 Κοζάνη      | ∅          | ∅
(11 rows)
```

Το `LEFT JOIN LATERAL ... ON true` κρατά την Κοζάνη με ∅. Με `CROSS JOIN LATERAL` θα χανόταν.

Άσκηση 14.7 · Η τελευταία κριτική

Για κάθε προϊόν με τουλάχιστον δύο κριτικές εμφάνισε όνομα, τη βαθμολογία και τον τίτλο της πιο πρόσφατης κριτικής του.

ΛΥΣΗ

```
SELECT p.name, last_r.rating, last_r.title
FROM products AS p
CROSS JOIN LATERAL (SELECT r.rating, r.title
                    FROM reviews AS r
                    WHERE r.product_id = p.id
                    ORDER BY r.created_at DESC
                    LIMIT 1) AS last_r
WHERE (SELECT count(*) FROM reviews AS r WHERE r.product_id = p.id) >= 2
ORDER BY p.name;
```

ΑΠΟΤΕΛΕΣΜΑ

```
                name                 | rating |       title
-------------------------------------+--------+-------------------
 Ακουστικά ANC                       |      3 | Ok
 Ασύρματο ποντίκι                    |      1 | Χάλασε
 Βίος και πολιτεία του Αλέξη Ζορμπά |      5 | Ζορμπάς για πάντα
 Εργονομική καρέκλα                  |      4 | Άνετη
 Καφετιέρα espresso                  |      2 | Διαρροή
 Μαθαίνω Python                      |      4 | Πρακτικό
 Μηχανικό πληκτρολόγιο               |      4 | θορυβώδες
 Οθόνη 27" 4K                        |      3 | Μέτρια
 Laptop Aero 14                      |      2 | Ζεσταίνεται
 Laptop Pro 16                       |      3 | Ακριβό
 Laptop Student 15                   |      2 | Αργό
 PostgreSQL σε βάθος                 |      5 | Must have
(12 rows)
```

Το `WHERE` με correlated subquery φιλτράρει τα προϊόντα και το `LATERAL` φέρνει την τελευταία κριτική. Τα δύο εργαλεία του κεφαλαίου συνδυάζονται χωρίς πρόβλημα στο ίδιο query.

ΚΕΦΑΛΑΙΟ 15

INSERT, UPDATE, DELETE και RETURNING

Άσκηση 15.1 · Νέα πόλη

Πρόσθεσε την πόλη Κέρκυρα με `id` 13, περιφέρεια `Ιόνια Νησιά` και πληθυσμό 32095. Επέστρεψε ολόκληρη τη γραμμή.

ΛΥΣΗ

```
INSERT INTO cities (id, name, region, population)
VALUES (13, 'Κέρκυρα', 'Ιόνια Νησιά', 32095)
RETURNING *;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id |  name   |   region    | population
----+---------+-------------+------------
 13 | Κέρκυρα | Ιόνια Νησιά |      32095
(1 row)

INSERT 0 1
```

Ο πίνακας `cities` δεν έχει identity column, οπότε το `id` δίνεται ρητά.

Άσκηση 15.2 · Νέος πελάτης με newsletter

Πρόσθεσε πελάτη με όνομα `Ορέστης`, επώνυμο `Δανιήλ`, email `orestis.d@example.gr`, πόλη 7, εγγραφή στο newsletter και ημερομηνία εγγραφής `2026-06-30 09:00`. Επέστρεψε το `id` και το `created_at`.

ΛΥΣΗ

```
INSERT INTO customers (first_name, last_name, email, city_id, newsletter, created_at)
VALUES ('Ορέστης', 'Δανιήλ', 'orestis.d@example.gr', 7, true, '2026-06-30 09:00')
RETURNING id, created_at;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id |       created_at
----+------------------------
 33 | 2026-06-30 09:00:00+03
(1 row)

INSERT 0 1
```

Το `id` είναι 33. Ο πελάτης της θεωρίας πήρε το 31 και το `INSERT` που απέτυχε με την πόλη 99 κατανάλωσε το 32. Η ακολουθία του identity δεν ξαναδίνει έναν αριθμό, ακόμη κι αν η εντολή που τον πήρε αποτύχει ή αναιρεθεί. Τα κενά στα `id` είναι φυσιολογικά και δεν σημαίνουν ότι χάθηκαν δεδομένα.

Άσκηση 15.3 · Νέο προϊόν με κατηγορία από όνομα

Πρόσθεσε το προϊόν με `sku` `AU-005`, όνομα `Ακουστικά gaming`, τιμή 89, κόστος 50 και απόθεμα 12, στην κατηγορία με όνομα `Ήχος`. Βρες το `category_id` με `INSERT ... SELECT` και όχι γράφοντας τον αριθμό.

ΛΥΣΗ

```
INSERT INTO products (sku, name, category_id, price, cost, stock)
SELECT 'AU-005', 'Ακουστικά gaming', id, 89, 50, 12
FROM categories
WHERE name = 'Ήχος'
RETURNING id, sku, category_id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id |  sku   | category_id
----+--------+-------------
 32 | AU-005 |          10
(1 row)

INSERT 0 1
```

Το `SELECT` επιστρέφει μία γραμμή, αφού τα ονόματα κατηγοριών είναι μοναδικά. Αν δεν υπήρχε κατηγορία `Ήχος`, δεν θα έμπαινε καμία γραμμή και το `psql` θα έγραφε `INSERT 0 0`.

Άσκηση 15.4 · Εκπτώσεις στην κουζίνα

Μείωσε κατά 15% τις τιμές των προϊόντων της κατηγορίας Κουζίνα, στρογγυλεμένες σε δύο δεκαδικά. Επέστρεψε το όνομα, την παλιά και τη νέα τιμή.

ΛΥΣΗ

```
UPDATE products
SET price = round(price * 0.85, 2)
WHERE category_id = 12
RETURNING name, old.price AS old_price, new.price AS new_price;
```

ΑΠΟΤΕΛΕΣΜΑ

```
        name        | old_price | new_price
--------------------+-----------+-----------
 Καφετιέρα espresso |    249.00 |    211.65
 Βραστήρας          |     34.90 |     29.67
 Σετ μαχαιριών      |     64.00 |     54.40
(3 rows)

UPDATE 3
```

Το `old` και το `new` είναι διαθέσιμα από την PostgreSQL 18. Σε παλαιότερες εκδόσεις το `RETURNING price` δίνει μόνο τη νέα τιμή.

Άσκηση 15.5 · Παράδοση των παλιών αποστολών

Άλλαξε σε `delivered` την κατάσταση των παραγγελιών που είναι `shipped` και στάλθηκαν πριν από τις 20 Ιουνίου 2026. Επέστρεψε `id` και `shipped_at`.

ΛΥΣΗ

```
UPDATE orders
SET status = 'delivered'
WHERE status = 'shipped'
  AND shipped_at < '2026-06-20'
RETURNING id, shipped_at;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id  |       shipped_at
-----+------------------------
 140 | 2026-06-11 10:09:00+03
 142 | 2026-06-17 18:32:00+03
 143 | 2026-06-16 21:35:00+03
 144 | 2026-06-18 11:46:00+03
 145 | 2026-06-15 22:04:00+03
 147 | 2026-06-16 02:41:00+03
 148 | 2026-06-18 12:47:00+03
 149 | 2026-06-19 16:05:00+03
 150 | 2026-06-17 00:06:00+03
(9 rows)

UPDATE 9
```

Η παραγγελία 150 είναι αντικαταβολή που μόλις παραδόθηκε. Σε ένα πραγματικό σύστημα η αλλαγή κατάστασης θα έπρεπε να συνοδεύεται από την πληρωμή της. Αυτό είναι δουλειά για transaction.

Άσκηση 15.6 · Ανανέωση αποθέματος από παραγγελία

Η παραγγελία 157 ακυρώνεται. Αύξησε το απόθεμα κάθε προϊόντος της κατά την ποσότητα που είχε παραγγελθεί. Επέστρεψε όνομα και νέο απόθεμα.

ΛΥΣΗ

```
UPDATE products AS p
SET stock = p.stock + oi.quantity
FROM order_items AS oi
WHERE oi.product_id = p.id
  AND oi.order_id = 157
RETURNING p.name, p.stock;
```

ΑΠΟΤΕΛΕΣΜΑ

```
      name       | stock
-----------------+-------
 Ματωμένα χώματα |     1
 Σετ μαχαιριών   |    17
(2 rows)

UPDATE 2
```

Κάθε προϊόν εμφανίζεται μία φορά σε μια παραγγελία, αφού το primary key του `order_items` είναι το ζεύγος παραγγελίας και προϊόντος. Γι' αυτό το `UPDATE ... FROM` εδώ είναι ασφαλές.

Άσκηση 15.7 · Διαγραφή πελάτη

Διέγραψε τον πελάτη 30. Πρώτα πρέπει να φύγουν τα events του, αλλιώς το foreign key θα εμποδίσει τη διαγραφή. Γράψε τις δύο εντολές με τη σωστή σειρά και επέστρεψε από τη δεύτερη το όνομα του πελάτη.

ΛΥΣΗ

```
DELETE FROM events WHERE customer_id = 30;
DELETE FROM customers WHERE id = 30 RETURNING first_name, last_name;
```

ΑΠΟΤΕΛΕΣΜΑ

```
DELETE 0

 first_name | last_name
------------+-----------
 Όλγα       | Βλάχου
(1 row)

DELETE 1
```

Αν έγραφες πρώτα το `DELETE FROM customers`, θα αποτύγχανε με παραβίαση του foreign key του `events`. Ο πελάτης 30 δεν έχει παραγγελίες ούτε κριτικές, αλλιώς θα χρειάζονταν κι αυτές. Το κεφάλαιο 17 δείχνει το `ON DELETE CASCADE`, που κάνει αυτή τη δουλειά αυτόματα. Εξηγεί επίσης γιατί δεν το θέλεις πάντα.

Άσκηση 15.8 · Αντίγραφο αρχείου

Διέγραψε τις κριτικές με βαθμολογία 2 που γράφτηκαν πριν από το 2026 και επέστρεψε όλες τις στήλες τους, ώστε να μπορείς να τις κρατήσεις κάπου αλλού.

ΛΥΣΗ

```
DELETE FROM reviews
WHERE rating = 2 AND created_at < '2026-01-01'
RETURNING id, product_id, customer_id, rating, title, created_at;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id | product_id | customer_id | rating |    title    |       created_at
----+------------+-------------+--------+-------------+------------------------
 11 |         22 |           8 |      2 | Διαρροή     | 2025-03-27 22:15:00+02
 30 |         13 |          13 |      2 | Αργό        | 2025-09-13 09:50:00+03
 36 |         11 |           9 |      2 | Ζεσταίνεται | 2025-11-01 14:33:00+02
(3 rows)

DELETE 3
```

Το `RETURNING` σε ένα `DELETE` είναι ο πιο απλός τρόπος να κρατήσεις αντίγραφο αυτού που διαγράφεις. Με `RETURNING *` θα έπαιρνες και το κείμενο της κριτικής. Στο κεφάλαιο 21 θα δεις πώς το αποτέλεσμα ενός `DELETE ... RETURNING` μπαίνει απευθείας σε άλλον πίνακα με ένα μόνο query.

ΚΕΦΑΛΑΙΟ 16

Upsert με ON CONFLICT και MERGE

Άσκηση 16.1 · Εισαγωγή χωρίς διπλά

Πρόσθεσε τις πόλεις Χανιά (8) και Άρτα (14, Ήπειρος, 43166) με ένα `INSERT`, αγνοώντας όσες υπάρχουν ήδη. Επέστρεψε τις γραμμές που μπήκαν.

ΛΥΣΗ

```
INSERT INTO cities (id, name, region, population)
VALUES (8, 'Χανιά', 'Κρήτη', 108642),
       (14, 'Άρτα', 'Ήπειρος', 43166)
ON CONFLICT DO NOTHING
RETURNING *;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id | name | region  | population
----+------+---------+------------
 14 | Άρτα | Ήπειρος |      43166
(1 row)

INSERT 0 1
```

Το `RETURNING` επιστρέφει μόνο την Άρτα. Τα Χανιά συγκρούστηκαν με το primary key και αγνοήθηκαν σιωπηλά.

Άσκηση 16.2 · Ενημέρωση επαφής από εισαγωγή

Ένα αρχείο φέρνει δύο πελάτες με email. Ο `maria.p@example.gr` υπάρχει ήδη και πρέπει να πάρει νέο τηλέφωνο `6900000001`. Ο `lina.k@example.gr` είναι νέος, με όνομα `Λίνα Καλλέργη` και τηλέφωνο `6900000002`. Χρησιμοποίησε ημερομηνία εγγραφής `2026-06-30`. Επέστρεψε `id`, email, αν η γραμμή είναι νέα και το νέο τηλέφωνο.

ΛΥΣΗ

```
INSERT INTO customers (first_name, last_name, email, phone, created_at)
VALUES ('Μαρία', 'Παπαδοπούλου', 'maria.p@example.gr', '6900000001', '2026-06-30'),
       ('Λίνα', 'Καλλέργη', 'lina.k@example.gr', '6900000002', '2026-06-30')
ON CONFLICT (email) DO UPDATE SET phone = excluded.phone
RETURNING id, email, old.id IS NULL AS inserted, new.phone;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id |       email        | inserted |   phone
----+--------------------+----------+------------
  1 | maria.p@example.gr | f        | 6900000001
 32 | lina.k@example.gr  | t        | 6900000002
(2 rows)

INSERT 0 2
```

Το `old.id IS NULL` είναι αληθές μόνο για τη γραμμή που μπήκε. Η νέα γραμμή πήρε `id` 32 και όχι 31. Το `INSERT` πήρε αριθμό από την ακολουθία και για την πρώτη γραμμή, πριν ανακαλύψει τη σύγκρουση. Ο υπάρχων πελάτης κρατά το αρχικό `created_at`, αφού το `DO UPDATE` αλλάζει μόνο το τηλέφωνο.

Άσκηση 16.3 · Μόνο ανατιμήσεις

Ο προμηθευτής 1 στέλνει νέες τιμές για τα προϊόντα 1, 2 και 3, 9.50, 7.50 και 4.60. Κάνε upsert αλλά ενημέρωσε μια υπάρχουσα τιμή μόνο αν η νέα είναι μεγαλύτερη. Επέστρεψε `product_id` και νέα τιμή.

ΛΥΣΗ

```
INSERT INTO product_suppliers (product_id, supplier_id, supply_price)
VALUES (1, 1, 9.50), (2, 1, 7.50), (3, 1, 4.60)
ON CONFLICT (product_id, supplier_id)
DO UPDATE SET supply_price = excluded.supply_price
WHERE excluded.supply_price > product_suppliers.supply_price
RETURNING product_id, new.supply_price;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 product_id | supply_price
------------+--------------
          1 |         9.50
          3 |         4.60
(2 rows)

INSERT 0 2
```

Το προϊόν 2 είχε τιμή 7.80 και η νέα 7.50 είναι μικρότερη, οπότε δεν ενημερώθηκε και δεν εμφανίζεται στο `RETURNING`.

Άσκηση 16.4 · Συγχρονισμός τιμοκαταλόγου

Με `MERGE`, συγχρόνισε τον `products` με τη λίστα `('PR-001', 22.90)`, `('PR-002', 89.00)` και `('PR-006', 19.90)`. Για υπάρχοντα `sku` ενημέρωσε την τιμή μόνο αν διαφέρει. Για νέα `sku` πρόσθεσε προϊόν με όνομα `Mousepad XL`, κατηγορία 9 και απόθεμα 50. Επέστρεψε ενέργεια, `sku` και τιμή.

ΛΥΣΗ

```
MERGE INTO products AS t
USING (VALUES ('PR-001', 22.90), ('PR-002', 89.00), ('PR-006', 19.90)) AS s(sku, price)
ON t.sku = s.sku
WHEN MATCHED AND t.price <> s.price THEN
  UPDATE SET price = s.price
WHEN NOT MATCHED THEN
  INSERT (sku, name, category_id, price, stock)
  VALUES (s.sku, 'Mousepad XL', 9, s.price, 50)
RETURNING merge_action() AS action, t.sku, t.price;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 action |  sku   | price
--------+--------+-------
 UPDATE | PR-001 | 22.90
 INSERT | PR-006 | 19.90
(2 rows)

MERGE 2
```

Το `PR-002` ταίριαξε αλλά η τιμή του είναι ήδη 89, οπότε η συνθήκη του `WHEN MATCHED` απέτυχε και δεν έγινε τίποτα. Γραμμές χωρίς ενέργεια δεν εμφανίζονται στο `RETURNING`.

Άσκηση 16.5 · Πλήρης λίστα προμηθευτή

Ο προμηθευτής 3 στέλνει την πλήρη λίστα του, `(16, 245.00)`, `(19, 118.00)` και `(20, 41.00)`. Με `MERGE`, ενημέρωσε τις υπάρχουσες τιμές, πρόσθεσε τα καινούρια και διέγραψε τα προϊόντα του προμηθευτή 3 που δεν είναι στη λίστα. Επέστρεψε ενέργεια και `product_id`, ταξινομημένα.

ΛΥΣΗ

```
MERGE INTO product_suppliers AS t
USING (VALUES (16, 245.00), (19, 118.00), (20, 41.00)) AS s(product_id, supply_price)
ON t.product_id = s.product_id AND t.supplier_id = 3
WHEN MATCHED THEN
  UPDATE SET supply_price = s.supply_price
WHEN NOT MATCHED THEN
  INSERT (product_id, supplier_id, supply_price) VALUES (s.product_id, 3, s.supply_price)
WHEN NOT MATCHED BY SOURCE AND t.supplier_id = 3 THEN
  DELETE
RETURNING merge_action() AS action, t.product_id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 action | product_id
--------+------------
 UPDATE |         16
 DELETE |         18
 UPDATE |         19
 UPDATE |         20
 DELETE |         21
 DELETE |         30
(6 rows)

MERGE 6
```

Η συνθήκη του `ON` περιέχει και το `t.supplier_id = 3`, αλλιώς το προϊόν 19 θα ταίριαζε και με τη γραμμή του προμηθευτή 2. Η ίδια συνθήκη στο `NOT MATCHED BY SOURCE` προστατεύει τους άλλους προμηθευτές από τη διαγραφή.

Άσκηση 16.6 · Upsert χωρίς unique

Δοκίμασε να κάνεις upsert στον `reviews` με `ON CONFLICT (product_id, customer_id) DO NOTHING`, για μια δεύτερη κριτική του πελάτη 1 στο προϊόν 19. Εξήγησε το σφάλμα.

ΛΥΣΗ

```
INSERT INTO reviews (product_id, customer_id, rating, title, body, created_at)
VALUES (19, 1, 4, 'Ξανά', 'Δεύτερη γνώμη.', '2026-06-30')
ON CONFLICT (product_id, customer_id) DO NOTHING;
```

ΑΠΟΤΕΛΕΣΜΑ

```
ERROR:  there is no unique or exclusion constraint matching the ON CONFLICT specification
```

Ο `reviews` δεν έχει unique constraint στο ζεύγος προϊόντος και πελάτη, οπότε ένας πελάτης μπορεί να γράψει πολλές κριτικές για το ίδιο προϊόν. Το `ON CONFLICT` δεν έχει σε τι να στηριχτεί. Αν ο κανόνας ήταν «μία κριτική ανά πελάτη και προϊόν», θα έπρεπε πρώτα να προστεθεί το constraint, κάτι που θα δούμε στο επόμενο κεφάλαιο.

ΚΕΦΑΛΑΙΟ 17

CREATE TABLE, τύποι και constraints

Άσκηση 17.1 · Πίνακας καταστημάτων

Φτιάξε πίνακα `stores` με `id` identity που δεν δέχεται τιμή από τον χρήστη, `name` κείμενο υποχρεωτικό και μοναδικό, `city_id` που δείχνει στον `cities`, `opened_on` ημερομηνία υποχρεωτική και `is_open` boolean με προεπιλογή `true`. Πρόσθεσε το κατάστημα `Κέντρο Αθήνας` στην πόλη 1 με ημερομηνία έναρξης `2024-03-01` και επέστρεψε ολόκληρη τη γραμμή.

ΛΥΣΗ

```
CREATE TABLE stores (
  id        integer GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  name      text NOT NULL UNIQUE,
  city_id   integer REFERENCES cities (id),
  opened_on date NOT NULL,
  is_open   boolean NOT NULL DEFAULT true
);

INSERT INTO stores (name, city_id, opened_on)
VALUES ('Κέντρο Αθήνας', 1, '2024-03-01')
RETURNING *;
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE TABLE

 id |     name      | city_id | opened_on  | is_open
----+---------------+---------+------------+---------
  1 | Κέντρο Αθήνας |       1 | 2024-03-01 | t
(1 row)

INSERT 0 1
```

Το `CREATE TABLE` και το `INSERT` εκτελούνται στο ίδιο transaction. Στην PostgreSQL και οι εντολές που αλλάζουν το σχήμα συμμετέχουν σε transactions, κάτι που θα αξιοποιήσουμε στο κεφάλαιο 19.

Άσκηση 17.2 · Κόστος μικρότερο από τιμή

Πρόσθεσε στον `products` constraint με όνομα `products_cost_below_price` που απαιτεί το κόστος να μην ξεπερνά την τιμή. Μετά δοκίμασε να αλλάξεις το κόστος του προϊόντος 1 σε 20.

ΛΥΣΗ

```
ALTER TABLE products
  ADD CONSTRAINT products_cost_below_price CHECK (cost <= price);
UPDATE products SET cost = 20 WHERE id = 1;
```

ΑΠΟΤΕΛΕΣΜΑ

```
ALTER TABLE

ERROR:  new row for relation "products" violates check constraint "products_cost_below_price"
DETAIL:  Failing row contains (1, BK-LIT-001, Βίος και πολιτεία του Αλέξη Ζορμπά, 2, 16.90,
20.00, 42, t, {βιβλίο,λογοτεχνία,bestseller}, {"pages": 432, "language": "el", "publisher":
"Καζαντζά..., Το μυθιστόρημα του Νίκου Καζαντζάκ..., 24.00).
```

Το `ALTER TABLE` πέτυχε, γιατί καμία υπάρχουσα γραμμή δεν παραβιάζει τον κανόνα. Η δωροκάρτα με `cost` `NULL` περνά, αφού το `CHECK` αγνοεί το άγνωστο. Το `UPDATE` απέτυχε με το όνομα του constraint στο μήνυμα.

Άσκηση 17.3 · Διπλές κριτικές

Βρες τα ζεύγη προϊόντος και πελάτη με περισσότερες από μία κριτική. Κράτα την πιο πρόσφατη κριτική κάθε ζεύγους, διέγραψε τις υπόλοιπες και πρόσθεσε το constraint `reviews_one_per_customer`. Επέστρεψε τα `id` που διαγράφηκαν.

ΛΥΣΗ

```
DELETE FROM reviews AS r
WHERE EXISTS (SELECT 1
              FROM reviews AS newer
              WHERE newer.product_id = r.product_id
                AND newer.customer_id = r.customer_id
                AND newer.created_at > r.created_at)
RETURNING r.id, r.product_id, r.customer_id;

ALTER TABLE reviews
  ADD CONSTRAINT reviews_one_per_customer UNIQUE (product_id, customer_id);
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id | product_id | customer_id
----+------------+-------------
 25 |         11 |           2
(1 row)

DELETE 1

ALTER TABLE
```

Μια κριτική διαγράφεται αν υπάρχει νεότερη του ίδιου πελάτη για το ίδιο προϊόν. Είναι anti join με `EXISTS` πάνω στον ίδιο πίνακα. Μετά τον καθαρισμό το `ALTER TABLE` περνά.

Άσκηση 17.4 · Newsletter με SET NULL

Φτιάξε πίνακα `newsletter_sends` με `id` identity, `customer_id` που δείχνει στον `customers` με `ON DELETE SET NULL` και `sent_at timestamptz` υποχρεωτικό. Βάλε δύο αποστολές στον πελάτη 26 στις `2026-06-01 09:00` και `2026-06-15 09:00`. Διέγραψε τον πελάτη 26 και εμφάνισε τον πίνακα.

ΛΥΣΗ

```
CREATE TABLE newsletter_sends (
  id          integer GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
  customer_id integer REFERENCES customers (id) ON DELETE SET NULL,
  sent_at     timestamptz NOT NULL
);

INSERT INTO newsletter_sends (customer_id, sent_at)
VALUES (26, '2026-06-01 09:00'), (26, '2026-06-15 09:00');

DELETE FROM events WHERE customer_id = 26;
DELETE FROM customers WHERE id = 26;

SELECT * FROM newsletter_sends ORDER BY id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE TABLE

INSERT 0 2

DELETE 20

DELETE 1

 id | customer_id |        sent_at
----+-------------+------------------------
  1 |           ∅ | 2026-06-01 09:00:00+03
  2 |           ∅ | 2026-06-15 09:00:00+03
(2 rows)
```

Οι αποστολές έμειναν ως ιστορικό με άγνωστο παραλήπτη. Τα events του πελάτη έπρεπε να διαγραφούν πρώτα, γιατί το foreign key του `events` δεν έχει `ON DELETE`. Η στήλη `customer_id` του `newsletter_sends` πρέπει να επιτρέπει `NULL`, αλλιώς το `SET NULL` θα αποτύγχανε.

Άσκηση 17.5 · Υπολογισμένο περιθώριο

Πρόσθεσε στον `products` generated column `margin` ίση με `price - cost`. Εμφάνισε όνομα, τιμή και `margin` για τα τρία προϊόντα με το μεγαλύτερο περιθώριο.

ΛΥΣΗ

```
ALTER TABLE products
  ADD COLUMN margin numeric GENERATED ALWAYS AS (price - cost);

SELECT name, price, margin
FROM products
WHERE margin IS NOT NULL
ORDER BY margin DESC
LIMIT 3;
```

ΑΠΟΤΕΛΕΣΜΑ

```
ALTER TABLE

       name        |  price  | margin
-------------------+---------+--------
 Laptop Pro 16     | 2199.00 | 479.00
 Laptop Aero 14    | 1149.00 | 259.00
 Laptop Classic 13 |  799.00 | 189.00
(3 rows)
```

Χωρίς το `WHERE margin IS NOT NULL` η δωροκάρτα θα εμφανιζόταν πρώτη, επειδή το `NULL` ταξινομείται ως μεγαλύτερο σε φθίνουσα σειρά. Το κεφάλαιο 4 το είχε προειδοποιήσει.

Άσκηση 17.6 · Κουπόνι μίας χρήσης ανά πελάτη

Φτιάξε πίνακα `coupon_redemptions` με `coupon_code` που δείχνει στον `coupons`, `customer_id` που δείχνει στον `customers`, `order_id` που δείχνει στον `orders` και είναι μοναδικό. Πρόσθεσε και κανόνα ότι ο ίδιος πελάτης δεν εξαργυρώνει το ίδιο κουπόνι δύο φορές. Βάλε μια εξαργύρωση του `WELCOME10` από τον πελάτη 1 στην παραγγελία 11 και δοκίμασε να βάλεις δεύτερη στην παραγγελία 17.

ΛΥΣΗ

```
CREATE TABLE coupon_redemptions (
  coupon_code text    NOT NULL REFERENCES coupons (code),
  customer_id integer NOT NULL REFERENCES customers (id),
  order_id    integer NOT NULL UNIQUE REFERENCES orders (id),
  PRIMARY KEY (coupon_code, customer_id)
);

INSERT INTO coupon_redemptions VALUES ('WELCOME10', 1, 11);
INSERT INTO coupon_redemptions VALUES ('WELCOME10', 1, 17);
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE TABLE

INSERT 0 1

ERROR:  duplicate key value violates unique constraint "coupon_redemptions_pkey"
DETAIL:  Key (coupon_code, customer_id)=(WELCOME10, 1) already exists.
```

Το primary key στο ζεύγος κουπονιού και πελάτη είναι ο κανόνας. Το `UNIQUE` στο `order_id` λέει ότι μια παραγγελία εξαργυρώνει το πολύ ένα κουπόνι. Στα δεδομένα μας ο πελάτης 1 χρησιμοποίησε το `WELCOME10` δύο φορές, κάτι που η βάση δεν θα επέτρεπε αν ο κανόνας υπήρχε από την αρχή.

ΚΕΦΑΛΑΙΟ 18

Σχεδίαση σχήματος και κανονικοποίηση

Άσκηση 18.1 · Εντοπισμός ασυνέπειας

Στον `flat_orders` της θεωρίας, βρες τις πόλεις που εμφανίζονται με περισσότερες από μία περιφέρειες. Εμφάνισε την πόλη και τις περιφέρειες σε μία στήλη.

ΛΥΣΗ

```
SELECT city, string_agg(DISTINCT region, ' · ') AS regions
FROM flat_orders
GROUP BY city
HAVING count(DISTINCT region) > 1;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 city  |         regions
-------+-------------------------
 Χανιά | Κρήτη · Κρήτη και Νησιά
(1 row)
```

Αυτό το query ελέγχει αν ισχύει η εξάρτηση της περιφέρειας από την πόλη. Σε ένα κανονικοποιημένο σχήμα δεν χρειάζεται, αφού η περιφέρεια υπάρχει μία φορά για κάθε πόλη.

Άσκηση 18.2 · Ανάκτηση του σχήματος

Από τον `flat_orders`, φτιάξε με `CREATE TABLE ... AS` έναν πίνακα `flat_cities` με μία γραμμή για κάθε ζεύγος πόλης και περιφέρειας. Εμφάνισέ τον ταξινομημένο κατά πόλη.

ΛΥΣΗ

```
CREATE TABLE flat_cities AS
SELECT DISTINCT city, region
FROM flat_orders;

SELECT * FROM flat_cities ORDER BY city, region;
```

ΑΠΟΤΕΛΕΣΜΑ

```
SELECT 11

    city     |       region
-------------+--------------------
 Αθήνα       | Αττική
 Βόλος       | Θεσσαλία
 Ηράκλειο    | Κρήτη
 Θεσσαλονίκη | Κεντρική Μακεδονία
 Ιωάννινα    | Ήπειρος
 Καλαμάτα    | Πελοπόννησος
 Λάρισα      | Θεσσαλία
 Πάτρα       | Δυτική Ελλάδα
 Ρόδος       | Νότιο Αιγαίο
 Χανιά       | Κρήτη
 Χανιά       | Κρήτη και Νησιά
(11 rows)
```

Ο νέος πίνακας έχει δύο γραμμές για τα Χανιά. Η διάσπαση ενός επίπεδου πίνακα φέρνει στην επιφάνεια τις ασυνέπειες που κουβαλούσε. Πριν προσθέσεις `UNIQUE (city)` πρέπει να αποφασίσεις ποια περιφέρεια είναι η σωστή.

Άσκηση 18.3 · Lookup table για καταστάσεις

Φτιάξε πίνακα `order_statuses` με `code` primary key, ελληνική ετικέτα `label` και `sort_order` ακέραιο μοναδικό. Γέμισέ τον με τις έξι καταστάσεις και σύνδεσε τη στήλη `orders.status` με foreign key. Μετά εμφάνισε πλήθος παραγγελιών ανά κατάσταση με την ελληνική ετικέτα, κατά `sort_order`.

ΛΥΣΗ

```
CREATE TABLE order_statuses (
  code       text PRIMARY KEY,
  label      text NOT NULL,
  sort_order integer NOT NULL UNIQUE
);

INSERT INTO order_statuses VALUES
  ('pending', 'σε αναμονή', 1), ('paid', 'πληρωμένη', 2),
  ('shipped', 'στον δρόμο', 3), ('delivered', 'παραδόθηκε', 4),
  ('cancelled', 'ακυρώθηκε', 5), ('refunded', 'επιστράφηκε', 6);

ALTER TABLE orders
  ADD CONSTRAINT orders_status_fkey FOREIGN KEY (status) REFERENCES order_statuses (code);

SELECT s.label, count(*) AS orders
FROM orders AS o
JOIN order_statuses AS s ON s.code = o.status
GROUP BY s.label, s.sort_order
ORDER BY s.sort_order;
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE TABLE

INSERT 0 6

ALTER TABLE

    label    | orders
-------------+--------
 σε αναμονή  |      3
 πληρωμένη   |      5
 στον δρόμο  |     12
 παραδόθηκε  |    125
 ακυρώθηκε   |     14
 επιστράφηκε |      3
(6 rows)
```

Το `CASE` του κεφαλαίου 5 για ελληνικές ετικέτες και σειρά ταξινόμησης έγινε δεδομένα. Μια νέα κατάσταση θα χρειαζόταν μόνο μία νέα γραμμή.

Άσκηση 18.4 · Πολλά τηλέφωνα

Φτιάξε πίνακα `customer_phones` με `customer_id`, `phone` και `kind` που δέχεται `κινητό`, `σταθερό` ή `εργασίας`. Κάθε τηλέφωνο να ανήκει σε ένα μόνο πελάτη. Μετέφερε τα υπάρχοντα τηλέφωνα των πελατών ως `κινητό` και εμφάνισε πόσα μεταφέρθηκαν.

ΛΥΣΗ

```
CREATE TABLE customer_phones (
  customer_id integer NOT NULL REFERENCES customers (id),
  phone       text NOT NULL UNIQUE,
  kind        text NOT NULL CHECK (kind IN ('κινητό', 'σταθερό', 'εργασίας')),
  PRIMARY KEY (customer_id, phone)
);

INSERT INTO customer_phones (customer_id, phone, kind)
SELECT id, phone, 'κινητό'
FROM customers
WHERE phone IS NOT NULL;

SELECT count(*) AS phones FROM customer_phones;
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE TABLE

INSERT 0 22

 phones
--------
     22
(1 row)
```

Ο νέος πίνακας είναι στην πρώτη κανονική μορφή. Ένας πελάτης μπορεί να έχει όσα τηλέφωνα θέλει και το `UNIQUE (phone)` εγγυάται ότι ένα τηλέφωνο δεν μοιράζεται σε δύο πελάτες. Στη στήλη `customers.phone` δεν μπορούσες να το εκφράσεις για δεύτερο τηλέφωνο.

Άσκηση 18.5 · Σύνοψη ως αντίγραφο

Φτιάξε με `CREATE TABLE ... AS` πίνακα `customer_stats` με `customer_id`, πλήθος παραγγελιών και ημερομηνία τελευταίας παραγγελίας. Εμφάνισε τη γραμμή του πελάτη 6. Μετά πρόσθεσε μια νέα παραγγελία στον πελάτη 6 και εμφάνισε ξανά τη γραμμή του.

ΛΥΣΗ

```
CREATE TABLE customer_stats AS
SELECT customer_id, count(*) AS orders, max(created_at)::date AS last_order
FROM orders
GROUP BY customer_id;

SELECT * FROM customer_stats WHERE customer_id = 6;

INSERT INTO orders (customer_id, status, created_at)
VALUES (6, 'pending', '2026-06-30 12:00');

SELECT * FROM customer_stats WHERE customer_id = 6;
```

ΑΠΟΤΕΛΕΣΜΑ

```
SELECT 27

 customer_id | orders | last_order
-------------+--------+------------
           6 |     10 | 2026-06-13
(1 row)

INSERT 0 1

 customer_id | orders | last_order
-------------+--------+------------
           6 |     10 | 2026-06-13
(1 row)
```

Ο `customer_stats` δεν άλλαξε. Είναι αντίγραφο της κατάστασης τη στιγμή που φτιάχτηκε. Αυτό είναι το κόστος του denormalization. Κάποιος πρέπει να τον ξαναϋπολογίσει ή να τον ενημερώνει. Τα materialized views του επόμενου κεφαλαίου κάνουν το πρώτο με μία εντολή.

ΚΕΦΑΛΑΙΟ 19

Transactions και savepoints

Άσκηση 19.1 · Παραγγελία που δεν έγινε

Σε transaction, πρόσθεσε μια παραγγελία `pending` του πελάτη 26 στις `2026-06-30 12:00`. Βάλε μια γραμμή με το προϊόν 3 και ποσότητα 2 στην τιμή του προϊόντος, χρησιμοποιώντας το `currval` της ακολουθίας των παραγγελιών για το `order_id`. Εμφάνισε τη γραμμή, κάνε `ROLLBACK` και δείξε ότι ο πελάτης 26 δεν έχει παραγγελίες.

ΛΥΣΗ

```
BEGIN;
INSERT INTO orders (customer_id, status, created_at)
VALUES (26, 'pending', '2026-06-30 12:00');
INSERT INTO order_items (order_id, product_id, quantity, unit_price)
SELECT currval(pg_get_serial_sequence('orders', 'id')), id, 2, price
FROM products WHERE id = 3;
SELECT order_id, product_id, quantity, unit_price FROM order_items
WHERE order_id = currval(pg_get_serial_sequence('orders', 'id'));
ROLLBACK;
SELECT count(*) AS orders FROM orders WHERE customer_id = 26;
```

ΑΠΟΤΕΛΕΣΜΑ

```
BEGIN

INSERT 0 1

INSERT 0 1

 order_id | product_id | quantity | unit_price
----------+------------+----------+------------
      164 |          3 |        2 |       9.90
(1 row)

ROLLBACK

 orders
--------
      0
(1 row)
```

Το `currval` δίνει τον τελευταίο αριθμό που πήρε αυτή η σύνδεση από την ακολουθία. Δεν επηρεάζεται από άλλες συνδέσεις, οπότε είναι ασφαλές. Το `pg_get_serial_sequence` βρίσκει το όνομα της ακολουθίας πίσω από το identity column.

Άσκηση 19.2 · Transaction σε σφάλμα

Ξεκίνα transaction, αύξησε κατά 10% την τιμή του προϊόντος 5 και μετά δοκίμασε να βάλεις προϊόν με `sku` `BK-PRG-001`, που υπάρχει ήδη. Δοκίμασε ένα `SELECT` της τιμής, κάνε `ROLLBACK` και εμφάνισε ξανά την τιμή.

ΛΥΣΗ

```
BEGIN;
UPDATE products SET price = round(price * 1.10, 2) WHERE id = 5;
INSERT INTO products (sku, name, price) VALUES ('BK-PRG-001', 'Διπλό', 10);
SELECT price FROM products WHERE id = 5;
ROLLBACK;
SELECT price FROM products WHERE id = 5;
```

ΑΠΟΤΕΛΕΣΜΑ

```
BEGIN

UPDATE 1

ERROR:  duplicate key value violates unique constraint "products_sku_key"
DETAIL:  Key (sku)=(BK-PRG-001) already exists.

ERROR:  current transaction is aborted, commands ignored until end of transaction block

ROLLBACK

 price
-------
 32.00
(1 row)
```

Το `SELECT` μέσα στο transaction απορρίφθηκε. Μετά το `ROLLBACK` η τιμή είναι η αρχική, γιατί το `UPDATE` ακυρώθηκε μαζί με την αποτυχημένη εισαγωγή.

Άσκηση 19.3 · Φόρτωση με savepoints

Σε transaction, πρόσθεσε τρεις κατηγορίες μία μία, με `id` 14, 13 και 15 και ονόματα `Παιχνίδια`, `Εργαλεία` και `Κήπος`, όλες με γονέα την 11. Βάλε savepoint πριν από κάθε εισαγωγή, ώστε η αποτυχία της δεύτερης, που έχει υπάρχον `id`, να μην ακυρώσει τις άλλες. Κάνε `COMMIT` και εμφάνισε τις κατηγορίες με `id` από 13 και πάνω.

ΛΥΣΗ

```
BEGIN;
SAVEPOINT row1;
INSERT INTO categories (id, name, parent_id) VALUES (14, 'Παιχνίδια', 11);
SAVEPOINT row2;
INSERT INTO categories (id, name, parent_id) VALUES (13, 'Εργαλεία', 11);
ROLLBACK TO SAVEPOINT row2;
SAVEPOINT row3;
INSERT INTO categories (id, name, parent_id) VALUES (15, 'Κήπος', 11);
COMMIT;
SELECT * FROM categories WHERE id >= 13 ORDER BY id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
BEGIN

SAVEPOINT

INSERT 0 1

SAVEPOINT

ERROR:  duplicate key value violates unique constraint "categories_pkey"
DETAIL:  Key (id)=(13) already exists.

ROLLBACK

SAVEPOINT

INSERT 0 1

COMMIT

 id |   name    | parent_id
----+-----------+-----------
 13 | Γραφείο   |        11
 14 | Παιχνίδια |        11
 15 | Κήπος     |        11
(3 rows)
```

Η δεύτερη εισαγωγή απέτυχε με παραβίαση του primary key. Το `ROLLBACK TO SAVEPOINT row2` επανέφερε το transaction σε κατάσταση που δέχεται εντολές και οι άλλες δύο κατηγορίες έγιναν commit. Σε πραγματικό κώδικα φόρτωσης, η εφαρμογή θα έβαζε savepoint πριν από κάθε γραμμή και θα επέστρεφε σε αυτό μόνο όταν η γραμμή αποτύχει.

Άσκηση 19.4 · Αλλαγή σχήματος που αναιρείται

Σε transaction, πρόσθεσε στον `customers` στήλη `loyalty_points` ακέραιο με προεπιλογή 0 και δώσε 100 πόντους στον πελάτη 1. Εμφάνισε τους πόντους του, κάνε `ROLLBACK` και έλεγξε αν η στήλη υπάρχει ακόμη ρωτώντας τον κατάλογο `information_schema.columns`.

ΛΥΣΗ

```
BEGIN;
ALTER TABLE customers ADD COLUMN loyalty_points integer NOT NULL DEFAULT 0;
UPDATE customers SET loyalty_points = 100 WHERE id = 1;
SELECT id, loyalty_points FROM customers WHERE id = 1;
ROLLBACK;
SELECT count(*) AS columns_named_loyalty
FROM information_schema.columns
WHERE table_name = 'customers' AND column_name = 'loyalty_points';
```

ΑΠΟΤΕΛΕΣΜΑ

```
BEGIN

ALTER TABLE

UPDATE 1

 id | loyalty_points
----+----------------
  1 |            100
(1 row)

ROLLBACK

 columns_named_loyalty
-----------------------
                     0
(1 row)
```

Ο κατάλογος `information_schema` περιγράφει τους πίνακες και τις στήλες της βάσης με queries. Μετά το `ROLLBACK` δεν υπάρχει καμία στήλη με αυτό το όνομα.

Άσκηση 19.5 · Ανταλλαγή θέσεων

Στον πίνακα `shelf` της θεωρίας, πρόσθεσε το προϊόν 3 στη θέση 3. Μετά, σε ένα transaction με deferred constraint, κάνε το προϊόν 3 πρώτο, το 1 δεύτερο και το 2 τρίτο. Εμφάνισε τον πίνακα κατά θέση.

ΛΥΣΗ

```
INSERT INTO shelf VALUES (3, 3);
BEGIN;
SET CONSTRAINTS shelf_position_unique DEFERRED;
UPDATE shelf SET position = 1 WHERE product_id = 3;
UPDATE shelf SET position = 2 WHERE product_id = 1;
UPDATE shelf SET position = 3 WHERE product_id = 2;
COMMIT;
SELECT * FROM shelf ORDER BY position;
```

ΑΠΟΤΕΛΕΣΜΑ

```
INSERT 0 1

BEGIN

SET CONSTRAINTS

UPDATE 1

UPDATE 1

UPDATE 1

COMMIT

 product_id | position
------------+----------
          3 |        1
          1 |        2
          2 |        3
(3 rows)
```

Μετά το πρώτο `UPDATE` δύο γραμμές έχουν θέση 1. Με άμεσο έλεγχο θα αποτύγχανε. Με deferred ο έλεγχος γίνεται στο `COMMIT`, όταν οι τρεις θέσεις είναι ξανά διαφορετικές.

ΚΕΦΑΛΑΙΟ 20

Views και materialized views

Άσκηση 20.1 · Επαφές πελατών

Φτιάξε view `customer_contacts` με `id`, ονοματεπώνυμο ως `full_name`, όνομα πόλης ή `άγνωστη` ως `city` και το πρώτο διαθέσιμο από email και τηλέφωνο ως `contact`. Εμφάνισε τους πελάτες 8 έως 12 από το view.

ΛΥΣΗ

```
CREATE VIEW customer_contacts AS
SELECT c.id,
       c.first_name || ' ' || c.last_name AS full_name,
       COALESCE(t.name, 'άγνωστη') AS city,
       COALESCE(c.email, c.phone) AS contact
FROM customers AS c
LEFT JOIN cities AS t ON t.id = c.city_id;

SELECT * FROM customer_contacts WHERE id BETWEEN 8 AND 12 ORDER BY id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE VIEW

 id |      full_name       |   city   |          contact
----+----------------------+----------+---------------------------
  8 | Παναγιώτης Βασιλείου | Αθήνα    | p.vasileiou@example.gr
  9 | Σοφία Αλεξίου        | Βόλος    | sofia.alexiou@example.gr
 10 | Χρήστος Μιχαηλίδης   | άγνωστη  | christos.m@example.gr
 11 | Ιωάννα Παππά         | Ιωάννινα | ioanna.pappa@example.gr
 12 | Αλέξανδρος Οικονόμου | Αθήνα    | alex.oikonomou@example.gr
(5 rows)
```

Όποιος διαβάζει το view δεν χρειάζεται να ξέρει ότι η πόλη βρίσκεται σε άλλο πίνακα ή ότι το email μπορεί να λείπει.

Άσκηση 20.2 · Μέση αξία ανά κατάσταση

Χρησιμοποιώντας το view `order_totals` της θεωρίας, εμφάνισε για κάθε κατάσταση το πλήθος των παραγγελιών και τη μέση αξία τους με δύο δεκαδικά.

ΛΥΣΗ

```
SELECT status, count(*) AS orders, round(avg(total), 2) AS avg_total
FROM order_totals
GROUP BY status
ORDER BY avg_total DESC;
```

ΑΠΟΤΕΛΕΣΜΑ

```
  status   | orders | avg_total
-----------+--------+-----------
 paid      |      6 |    600.72
 pending   |      3 |    548.07
 refunded  |      3 |    539.67
 delivered |    125 |    478.62
 cancelled |     14 |    252.00
 shipped   |     12 |    167.06
(6 rows)
```

Το view έχει ήδη το `GROUP BY` της παραγγελίας. Το query ομαδοποιεί ξανά, σε άλλο επίπεδο, όπως έκανε το derived table του κεφαλαίου 14.

Άσκηση 20.3 · Εισαγωγή μέσα από view

Μέσα από το `active_products`, πρόσθεσε το προϊόν `PR-007` με όνομα `Hub USB`, κατηγορία 9, τιμή 29, απόθεμα 40 και `active` αληθές. Μετά δοκίμασε να προσθέσεις το `PR-008` με `active` ψευδές.

ΛΥΣΗ

```
INSERT INTO active_products (sku, name, category_id, price, stock, active)
VALUES ('PR-007', 'Hub USB', 9, 29, 40, true)
RETURNING id, sku;
INSERT INTO active_products (sku, name, category_id, price, stock, active)
VALUES ('PR-008', 'Καλώδιο HDMI', 9, 9, 100, false);
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id |  sku
----+--------
 31 | PR-007
(1 row)

INSERT 0 1

ERROR:  new row violates check option for view "active_products"
DETAIL:  Failing row contains (32, PR-008, Καλώδιο HDMI, 9, 9.00, null, 100, f, {}, {}, null).
```

Η πρώτη εισαγωγή πέρασε στον `products` μέσω του view. Η δεύτερη απορρίφθηκε από το `WITH CHECK OPTION`, αφού η γραμμή δεν θα εμφανιζόταν στο view.

Άσκηση 20.4 · Κορυφαία προϊόντα ως materialized view

Φτιάξε materialized view `product_sales` με `product_id`, όνομα, συνολικά τεμάχια και έσοδα από παραγγελίες που δεν ακυρώθηκαν ούτε επιστράφηκαν. Εμφάνισε τα τρία προϊόντα με τα μεγαλύτερα έσοδα.

ΛΥΣΗ

```
CREATE MATERIALIZED VIEW product_sales AS
SELECT p.id AS product_id, p.name,
       sum(oi.quantity) AS units,
       sum(oi.quantity * oi.unit_price) AS revenue
FROM order_items AS oi
JOIN orders AS o ON o.id = oi.order_id
JOIN products AS p ON p.id = oi.product_id
WHERE o.status NOT IN ('cancelled', 'refunded')
GROUP BY p.id;

SELECT name, units, revenue FROM product_sales ORDER BY revenue DESC LIMIT 3;
```

ΑΠΟΤΕΛΕΣΜΑ

```
SELECT 29

       name        | units | revenue
-------------------+-------+----------
 Laptop Pro 16     |    11 | 23989.00
 Laptop Student 15 |    11 |  7139.00
 Laptop Aero 14    |     6 |  6844.00
(3 rows)
```

Η θεωρία πρόσθεσε ένα `Laptop Pro 16` στον Ιούνιο, οπότε τα έσοδά του περιλαμβάνονται. Το materialized view τα κράτησε τη στιγμή που φτιάχτηκε.

Άσκηση 20.5 · Παλιά δεδομένα και ανανέωση

Στο `product_sales` της προηγούμενης άσκησης, πρόσθεσε μια πληρωμένη παραγγελία του πελάτη 26 με τρία `Ασύρματο ποντίκι`. Εμφάνισε τη γραμμή του προϊόντος από το materialized view πριν και μετά από `REFRESH`.

ΛΥΣΗ

```
CREATE MATERIALIZED VIEW product_sales AS
SELECT p.id AS product_id, p.name,
       sum(oi.quantity) AS units,
       sum(oi.quantity * oi.unit_price) AS revenue
FROM order_items AS oi
JOIN orders AS o ON o.id = oi.order_id
JOIN products AS p ON p.id = oi.product_id
WHERE o.status NOT IN ('cancelled', 'refunded')
GROUP BY p.id;

INSERT INTO orders (customer_id, status, created_at)
VALUES (26, 'paid', '2026-06-30 13:00');
INSERT INTO order_items (order_id, product_id, quantity, unit_price)
VALUES (currval(pg_get_serial_sequence('orders', 'id')), 14, 3, 24.90);

SELECT units, revenue FROM product_sales WHERE product_id = 14;
REFRESH MATERIALIZED VIEW product_sales;
SELECT units, revenue FROM product_sales WHERE product_id = 14;
```

ΑΠΟΤΕΛΕΣΜΑ

```
SELECT 29

INSERT 0 1

INSERT 0 1

 units | revenue
-------+---------
     9 |  224.10
(1 row)

REFRESH MATERIALIZED VIEW

 units | revenue
-------+---------
    12 |  298.80
(1 row)
```

Η λύση ξαναφτιάχνει το materialized view, επειδή κάθε άσκηση ξεκινά από την κατάσταση του τέλους της θεωρίας. Πριν από το `REFRESH` τα τρία ποντίκια λείπουν. Μετά εμφανίζονται στα τεμάχια και στα έσοδα.

ΚΕΦΑΛΑΙΟ 21

CTE με WITH

Άσκηση 21.1 · Πληρωμές ανά περιφέρεια

Με δύο CTE, υπολόγισε πρώτα το καθαρό ποσό που πλήρωσε κάθε πελάτης και μετά άθροισέ το ανά περιφέρεια κατοικίας. Εμφάνισε περιφέρεια, πλήθος πελατών με πληρωμές και σύνολο, από το μεγαλύτερο σύνολο.

ΛΥΣΗ

```
WITH customer_paid AS (
  SELECT o.customer_id, sum(pay.amount) AS paid
  FROM payments AS pay
  JOIN orders AS o ON o.id = pay.order_id
  GROUP BY o.customer_id
),
by_region AS (
  SELECT t.region, count(*) AS customers, sum(cp.paid) AS paid
  FROM customer_paid AS cp
  JOIN customers AS c ON c.id = cp.customer_id
  JOIN cities AS t ON t.id = c.city_id
  GROUP BY t.region
)
SELECT * FROM by_region ORDER BY paid DESC;
```

ΑΠΟΤΕΛΕΣΜΑ

```
       region       | customers |   paid
--------------------+-----------+----------
 Αττική             |         7 | 18233.60
 Κεντρική Μακεδονία |         6 | 15901.30
 Θεσσαλία           |         4 | 10167.30
 Κρήτη              |         5 |  9150.40
 Δυτική Ελλάδα      |         2 |  5092.60
 Ήπειρος            |         1 |  2101.80
 Πελοπόννησος       |         1 |  1158.60
 Νότιο Αιγαίο       |         1 |   662.50
(8 rows)
```

Οι πελάτες χωρίς πόλη χάνονται στο join με τον `cities`. Αν έπρεπε να μετρηθούν, θα έγραφες `LEFT JOIN` και `COALESCE(t.region, 'άγνωστη')`.

Άσκηση 21.2 · Πάνω από τον μέσο όρο της κατηγορίας τους

Με CTE, υπολόγισε τα έσοδα κάθε προϊόντος από παραγγελίες που δεν ακυρώθηκαν ούτε επιστράφηκαν και τον μέσο όρο των εσόδων ανά κατηγορία. Εμφάνισε τα προϊόντα με έσοδα πάνω από τον μέσο όρο της κατηγορίας τους.

ΛΥΣΗ

```
WITH product_revenue AS (
  SELECT p.id, p.name, p.category_id, sum(oi.quantity * oi.unit_price) AS revenue
  FROM order_items AS oi
  JOIN orders AS o ON o.id = oi.order_id
  JOIN products AS p ON p.id = oi.product_id
  WHERE o.status NOT IN ('cancelled', 'refunded')
  GROUP BY p.id
),
category_avg AS (
  SELECT category_id, avg(revenue) AS avg_revenue
  FROM product_revenue
  GROUP BY category_id
)
SELECT pr.name, pr.revenue, round(ca.avg_revenue, 2) AS category_avg
FROM product_revenue AS pr
JOIN category_avg AS ca ON ca.category_id = pr.category_id
WHERE pr.revenue > ca.avg_revenue
ORDER BY pr.category_id, pr.revenue DESC;
```

ΑΠΟΤΕΛΕΣΜΑ

```
                name                 | revenue  | category_avg
-------------------------------------+----------+--------------
 Βίος και πολιτεία του Αλέξη Ζορμπά |   253.50 |       192.78
 Ματωμένα χώματα                     |   228.00 |       192.78
 Μαθαίνω Python                      |   541.00 |       430.67
 Καθαρός κώδικας στην πράξη          |   456.00 |       430.67
 PostgreSQL σε βάθος                 |   585.00 |       562.50
 Laptop Pro 16                       | 21790.00 |      9143.00
 Οθόνη 27" 4K                        |  3948.00 |      1340.00
 Ακουστικά ANC                       |  2647.00 |      1661.67
 Μικρόφωνο USB                       |  1785.00 |      1661.67
 Καφετιέρα espresso                  |  2241.00 |      1173.63
 Εργονομική καρέκλα                  |  4624.00 |      2947.33
 Γραφείο ρυθμιζόμενου ύψους          |  3672.00 |      2947.33
(12 rows)
```

Είναι το ίδιο σχήμα με την άσκηση 14.2, όπου ο μέσος όρος υπολογιζόταν με correlated subquery. Εδώ υπολογίζεται μία φορά ανά κατηγορία και ενώνεται.

Άσκηση 21.3 · Παραγγελία σε ένα query

Με data modifying CTE, πρόσθεσε σε ένα query μια παραγγελία `pending` του πελάτη 26 στις `2026-06-30 14:00` με δύο γραμμές, τα προϊόντα 5 και 6 σε ποσότητα 1 και στην τρέχουσα τιμή τους. Επέστρεψε το `order_id`, το `product_id` και την τιμή των γραμμών.

ΛΥΣΗ

```
WITH new_order AS (
  INSERT INTO orders (customer_id, status, created_at)
  VALUES (26, 'pending', '2026-06-30 14:00')
  RETURNING id
)
INSERT INTO order_items (order_id, product_id, quantity, unit_price)
SELECT new_order.id, p.id, 1, p.price
FROM new_order
CROSS JOIN products AS p
WHERE p.id IN (5, 6)
RETURNING order_id, product_id, unit_price;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 order_id | product_id | unit_price
----------+------------+------------
      164 |          5 |      32.00
      164 |          6 |      29.50
(2 rows)

INSERT 0 2
```

Το `CROSS JOIN` με το CTE, που έχει μία γραμμή, φέρνει το νέο `id` δίπλα σε κάθε προϊόν. Δεν χρειάζεται `currval` ούτε δεύτερο query.

Άσκηση 21.4 · Αρχειοθέτηση κριτικών

Φτιάξε πίνακα `reviews_archive` με την ίδια δομή με τον `reviews`, με `CREATE TABLE reviews_archive (LIKE reviews)`. Μετακίνησε σε αυτόν με ένα query τις κριτικές που γράφτηκαν πριν από τον Ιούλιο του 2025 και επέστρεψε το πλήθος τους. Μετά εμφάνισε πόσες κριτικές έμειναν στον `reviews`.

ΛΥΣΗ

```
CREATE TABLE reviews_archive (LIKE reviews);

WITH moved AS (
  DELETE FROM reviews
  WHERE created_at < '2025-07-01'
  RETURNING *
)
INSERT INTO reviews_archive
SELECT * FROM moved;

SELECT (SELECT count(*) FROM reviews_archive) AS archived,
       (SELECT count(*) FROM reviews) AS remaining;
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE TABLE

INSERT 0 23

 archived | remaining
----------+-----------
       23 |        16
(1 row)
```

Η διαγραφή και η εισαγωγή είναι ένα query και ένα transaction. Αν η εισαγωγή αποτύγχανε, οι κριτικές δεν θα χάνονταν. Το `LIKE reviews` αντιγράφει στήλες και τύπους, όχι όμως constraints και defaults, εκτός αν το ζητήσεις με επιλογές όπως `INCLUDING ALL`.

Άσκηση 21.5 · Τι βλέπει το κύριο query

Σε ένα query με CTE, αύξησε κατά 5 το απόθεμα των προϊόντων της κατηγορίας 13 επιστρέφοντας `id` και νέο απόθεμα. Στο κύριο query εμφάνισε για κάθε προϊόν το όνομα από τον `products`, το απόθεμα όπως το βλέπει ο πίνακας και το απόθεμα από το `RETURNING`.

ΛΥΣΗ

```
WITH restocked AS (
  UPDATE products SET stock = stock + 5
  WHERE category_id = 13
  RETURNING id, stock
)
SELECT p.name, p.stock AS table_stock, r.stock AS returned_stock
FROM restocked AS r
JOIN products AS p ON p.id = r.id
ORDER BY p.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
            name             | table_stock | returned_stock
-----------------------------+-------------+----------------
 Εργονομική καρέκλα          |           6 |             11
 Γραφείο ρυθμιζόμενου ύψους  |           3 |              8
 Φωτιστικό γραφείου LED      |          27 |             32
(3 rows)
```

Ο `products` στο κύριο query βλέπει την εικόνα πριν από το `UPDATE`, οπότε το `table_stock` είναι μικρότερο κατά 5. Μετά το τέλος του query, κάθε νέο query βλέπει τη νέα τιμή.

ΚΕΦΑΛΑΙΟ 22

Recursive CTE για δέντρα και γράφους

Άσκηση 22.1 · Το υποδέντρο των ηλεκτρονικών

Εμφάνισε όλες τις κατηγορίες κάτω από τα Ηλεκτρονικά, μαζί με αυτήν, με το βάθος τους και ταξινομημένες κατά μονοπάτι.

ΛΥΣΗ

```
WITH RECURSIVE tree AS (
  SELECT id, name, name AS path, 0 AS depth
  FROM categories
  WHERE name = 'Ηλεκτρονικά'
  UNION ALL
  SELECT c.id, c.name, t.path || ' / ' || c.name, t.depth + 1
  FROM categories AS c
  JOIN tree AS t ON c.parent_id = t.id
)
SELECT id, name, depth, path
FROM tree
ORDER BY path;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id |     name     | depth |                    path
----+--------------+-------+-----------------------------------------
  6 | Ηλεκτρονικά  |     0 | Ηλεκτρονικά
 10 | Ήχος         |     1 | Ηλεκτρονικά / Ήχος
  7 | Υπολογιστές  |     1 | Ηλεκτρονικά / Υπολογιστές
  9 | Περιφερειακά |     2 | Ηλεκτρονικά / Υπολογιστές / Περιφερειακά
  8 | Laptops      |     2 | Ηλεκτρονικά / Υπολογιστές / Laptops
(5 rows)
```

Το anchor ξεκινά από την κατηγορία που μας ενδιαφέρει αντί για τις ρίζες. Ο αναδρομικός όρος είναι ακριβώς ίδιος με τη θεωρία.

Άσκηση 22.2 · Breadcrumb κάθε προϊόντος

Για κάθε ενεργό προϊόν της κατηγορίας 4 και της κατηγορίας 9 εμφάνισε το όνομά του και το πλήρες μονοπάτι της κατηγορίας του από τη ρίζα.

ΛΥΣΗ

```
WITH RECURSIVE tree AS (
  SELECT id, name AS path
  FROM categories
  WHERE parent_id IS NULL
  UNION ALL
  SELECT c.id, t.path || ' / ' || c.name
  FROM categories AS c
  JOIN tree AS t ON c.parent_id = t.id
)
SELECT p.name, t.path
FROM products AS p
JOIN tree AS t ON t.id = p.category_id
WHERE p.active AND p.category_id IN (4, 9)
ORDER BY t.path, p.name;
```

ΑΠΟΤΕΛΕΣΜΑ

```
            name            |                   path
----------------------------+-------------------------------------------
 Καθαρός κώδικας στην πράξη | Βιβλία / Τεχνολογία / Προγραμματισμός
 Μαθαίνω Python             | Βιβλία / Τεχνολογία / Προγραμματισμός
 Ruby από την αρχή          | Βιβλία / Τεχνολογία / Προγραμματισμός
 Ασύρματο ποντίκι           | Ηλεκτρονικά / Υπολογιστές / Περιφερειακά
 Μηχανικό πληκτρολόγιο      | Ηλεκτρονικά / Υπολογιστές / Περιφερειακά
 Οθόνη 27" 4K               | Ηλεκτρονικά / Υπολογιστές / Περιφερειακά
 USB-C docking station      | Ηλεκτρονικά / Υπολογιστές / Περιφερειακά
 Webcam HD                  | Ηλεκτρονικά / Υπολογιστές / Περιφερειακά
(8 rows)
```

Αντί να υπολογίσεις το breadcrumb χωριστά για κάθε προϊόν, το CTE φτιάχνει το μονοπάτι όλων των κατηγοριών μία φορά και το join το φέρνει δίπλα σε κάθε προϊόν.

Άσκηση 22.3 · Η ομάδα των Πωλήσεων

Βρες όλους τους υπαλλήλους που αναφέρονται άμεσα ή έμμεσα στον υπάλληλο 3. Εμφάνισε όνομα, θέση και επίπεδο, όπου 1 σημαίνει άμεση αναφορά.

ΛΥΣΗ

```
WITH RECURSIVE team AS (
  SELECT id, full_name, title, 1 AS level
  FROM employees
  WHERE manager_id = 3
  UNION ALL
  SELECT e.id, e.full_name, e.title, t.level + 1
  FROM employees AS e
  JOIN team AS t ON e.manager_id = t.id
)
SELECT full_name, title, level
FROM team
ORDER BY level, full_name;
```

ΑΠΟΤΕΛΕΣΜΑ

```
    full_name    |      title      | level
-----------------+-----------------+-------
 Ιωάννα Ρήγα     | Account Manager |     1
 Στέλιος Παππάς  | Account Manager |     1
 Βασιλική Ντόκου | Sales Intern    |     2
(3 rows)
```

Το anchor ξεκινά από τους άμεσους υφισταμένους και όχι από τον ίδιο τον υπάλληλο 3, οπότε αυτός δεν εμφανίζεται στο αποτέλεσμα.

Άσκηση 22.4 · Πόσους έχει από κάτω του κάθε προϊστάμενος

Για κάθε υπάλληλο που έχει τουλάχιστον έναν υφιστάμενο, εμφάνισε το όνομά του, πόσους έχει άμεσα και πόσους συνολικά σε όλα τα επίπεδα.

ΛΥΣΗ

```
WITH RECURSIVE chain AS (
  SELECT manager_id AS boss, id AS subordinate, 1 AS level
  FROM employees
  WHERE manager_id IS NOT NULL
  UNION ALL
  SELECT c.boss, e.id, c.level + 1
  FROM chain AS c
  JOIN employees AS e ON e.manager_id = c.subordinate
)
SELECT b.full_name,
       count(*) FILTER (WHERE c.level = 1) AS direct,
       count(*) AS total
FROM chain AS c
JOIN employees AS b ON b.id = c.boss
GROUP BY b.id
ORDER BY total DESC, b.full_name;
```

ΑΠΟΤΕΛΕΣΜΑ

```
       full_name       | direct | total
-----------------------+--------+-------
 Ελένη Μαυρίδου        |      3 |    13
 Νίκος Αλεξίου         |      3 |     4
 Δημήτρης Λαμπρόπουλος |      2 |     3
 Σοφία Κωνσταντίνου    |      2 |     3
 Γιάννης Κορρές        |      1 |     1
 Ιωάννα Ρήγα           |      1 |     1
 Κατερίνα Ζαφειρίου    |      1 |     1
(7 rows)
```

Κάθε γραμμή του `chain` είναι ένα ζεύγος προϊσταμένου και υφισταμένου σε οποιοδήποτε επίπεδο. Είναι το ίδιο μοτίβο με το `root_id` της θεωρίας, με τη διαφορά ότι ξεκινά από κάθε προϊστάμενο και όχι μόνο από τις ρίζες.

Άσκηση 22.5 · Έσοδα ανά κορυφαία κατηγορία

Υπολόγισε τα έσοδα από παραγγελίες που δεν ακυρώθηκαν ούτε επιστράφηκαν για κάθε μία από τις τρεις ρίζες του δέντρου, μαζί με όλες τις υποκατηγορίες τους.

ΛΥΣΗ

```
WITH RECURSIVE tree AS (
  SELECT id AS root_id, id
  FROM categories
  WHERE parent_id IS NULL
  UNION ALL
  SELECT t.root_id, c.id
  FROM categories AS c
  JOIN tree AS t ON c.parent_id = t.id
)
SELECT root.name AS root_category,
       sum(oi.quantity * oi.unit_price) AS revenue
FROM tree AS t
JOIN categories AS root ON root.id = t.root_id
JOIN products AS p ON p.category_id = t.id
JOIN order_items AS oi ON oi.product_id = p.id
JOIN orders AS o ON o.id = oi.order_id
WHERE o.status NOT IN ('cancelled', 'refunded')
GROUP BY root.name
ORDER BY revenue DESC;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 root_category | revenue
---------------+----------
 Ηλεκτρονικά   | 48232.10
 Σπίτι         | 12320.90
 Βιβλία        |  3680.10
(3 rows)
```

Η δωροκάρτα δεν ανήκει σε καμία κατηγορία, οπότε δεν μετράει σε καμία ρίζα. Η αλυσίδα των joins μετά το CTE είναι από «πολλά» προς «ένα» και προς «πολλά» μόνο μία φορά, οπότε δεν υπάρχει fan out.

Άσκηση 22.6 · Η συντομότερη διαδρομή

Με τον πίνακα `routes` της θεωρίας, βρες τη συντομότερη σε ώρες διαδρομή από την Αθήνα στη Θεσσαλονίκη. Εμφάνισε τις ώρες και τη διαδρομή με ονόματα πόλεων, όπως `Αθήνα / Λάρισα / Θεσσαλονίκη`.

ΛΥΣΗ

```
WITH RECURSIVE reach AS (
  SELECT r.to_city AS city, r.hours::numeric AS total_hours,
         'Αθήνα / ' || c.name AS route
  FROM routes AS r
  JOIN cities AS c ON c.id = r.to_city
  WHERE r.from_city = 1
  UNION ALL
  SELECT r.to_city, re.total_hours + r.hours, re.route || ' / ' || c.name
  FROM routes AS r
  JOIN reach AS re ON r.from_city = re.city
  JOIN cities AS c ON c.id = r.to_city
) CYCLE city SET is_cycle USING path
SELECT total_hours, route
FROM reach
WHERE city = 2 AND NOT is_cycle
ORDER BY total_hours
LIMIT 1;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 total_hours |            route
-------------+-----------------------------
         6.5 | Αθήνα / Λάρισα / Θεσσαλονίκη
(1 row)
```

Η αναδρομή βρίσκει όλες τις διαδρομές χωρίς κύκλο και το `ORDER BY` με `LIMIT 1` κρατά τη συντομότερη. Υπάρχει και διαδρομή μέσω Πάτρας και Ιωαννίνων, που είναι μεγαλύτερη. Σε μεγάλους γράφους αυτή η προσέγγιση εξερευνά πολλές διαδρομές. Για τέτοια προβλήματα υπάρχουν εξειδικευμένες επεκτάσεις, όπως η pgRouting.

ΚΕΦΑΛΑΙΟ 23

Window functions για κατάταξη και top N

Άσκηση 23.1 · Απόσταση από τον μέσο όρο

Για τα ενεργά προϊόντα των κατηγοριών 12 και 13 εμφάνισε όνομα, τιμή, μέση τιμή της κατηγορίας τους και τη διαφορά της τιμής από αυτήν, με δύο δεκαδικά.

ΛΥΣΗ

```
SELECT category_id, name, price,
       round(avg(price) OVER (PARTITION BY category_id), 2) AS category_avg,
       round(price - avg(price) OVER (PARTITION BY category_id), 2) AS diff
FROM products
WHERE active AND category_id IN (12, 13)
ORDER BY category_id, price DESC;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 category_id |            name            | price  | category_avg |  diff
-------------+----------------------------+--------+--------------+---------
          12 | Καφετιέρα espresso         | 249.00 |       115.97 |  133.03
          12 | Σετ μαχαιριών              |  64.00 |       115.97 |  -51.97
          12 | Βραστήρας                  |  34.90 |       115.97 |  -81.07
          13 | Γραφείο ρυθμιζόμενου ύψους | 459.00 |       263.33 |  195.67
          13 | Εργονομική καρέκλα         | 289.00 |       263.33 |   25.67
          13 | Φωτιστικό γραφείου LED     |  42.00 |       263.33 | -221.33
(6 rows)
```

Η window function χρησιμοποιείται μέσα σε αριθμητική έκφραση όπως κάθε άλλη τιμή.

Άσκηση 23.2 · Κατάταξη μισθών ανά τμήμα

Για κάθε υπάλληλο με τμήμα και μισθό εμφάνισε τμήμα, όνομα, μισθό και τη θέση του μισθού του μέσα στο τμήμα, από τον υψηλότερο, με `dense_rank`.

ΛΥΣΗ

```
SELECT department, full_name, salary,
       dense_rank() OVER (PARTITION BY department ORDER BY salary DESC) AS position
FROM employees
WHERE department IS NOT NULL AND salary IS NOT NULL
ORDER BY department, position;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 department |       full_name       | salary  | position
------------+-----------------------+---------+----------
 Αποθήκη    | Δημήτρης Λαμπρόπουλος | 6200.00 |        1
 Αποθήκη    | Γιάννης Κορρές        | 1800.00 |        2
 Αποθήκη    | Θανάσης Μπέης         | 1750.00 |        3
 Αποθήκη    | Χριστίνα Αθανασίου    | 1650.00 |        4
 Διοίκηση   | Ελένη Μαυρίδου        | 9500.00 |        1
 Πωλήσεις   | Σοφία Κωνσταντίνου    | 7000.00 |        1
 Πωλήσεις   | Ιωάννα Ρήγα           | 3100.00 |        2
 Πωλήσεις   | Στέλιος Παππάς        | 2900.00 |        3
 Πωλήσεις   | Βασιλική Ντόκου       | 1100.00 |        4
 Τεχνολογία | Νίκος Αλεξίου         | 8200.00 |        1
 Τεχνολογία | Μαρίνα Σταθοπούλου    | 4400.00 |        2
 Τεχνολογία | Κατερίνα Ζαφειρίου    | 4200.00 |        3
 Τεχνολογία | Άγγελος Φωτίου        | 3900.00 |        4
 Τεχνολογία | Πέτρος Χατζής         | 2100.00 |        5
(14 rows)
```

Στην Αποθήκη οι δύο αποθηκάριοι έχουν διαφορετικό μισθό, οπότε κάθε θέση έχει έναν υπάλληλο. Με ισοβαθμία το `dense_rank` θα έδινε την ίδια θέση και ο επόμενος δεν θα πηδούσε αριθμό.

Άσκηση 23.3 · Οι δύο καλύτεροι πελάτες κάθε περιφέρειας

Για κάθε περιφέρεια εμφάνισε τους δύο πελάτες με τα περισσότερα έσοδα από παραγγελίες που δεν ακυρώθηκαν ούτε επιστράφηκαν. Εμφάνισε περιφέρεια, θέση, όνομα και έσοδα.

ΛΥΣΗ

```
WITH spent AS (
  SELECT c.id, c.first_name, c.last_name, t.region,
         sum(oi.quantity * oi.unit_price) AS revenue
  FROM customers AS c
  JOIN cities AS t ON t.id = c.city_id
  JOIN orders AS o ON o.customer_id = c.id
  JOIN order_items AS oi ON oi.order_id = o.id
  WHERE o.status NOT IN ('cancelled', 'refunded')
  GROUP BY c.id, t.region
),
ranked AS (
  SELECT *, row_number() OVER (PARTITION BY region ORDER BY revenue DESC, id) AS pos
  FROM spent
)
SELECT region, pos, first_name, last_name, revenue
FROM ranked
WHERE pos <= 2
ORDER BY region, pos;
```

ΑΠΟΤΕΛΕΣΜΑ

```
       region       | pos | first_name |  last_name   | revenue
--------------------+-----+------------+--------------+---------
 Αττική             |   1 | Στέλιος    | Κουτσούκος   | 7291.60
 Αττική             |   2 | Παναγιώτης | Βασιλείου    | 4401.80
 Δυτική Ελλάδα      |   1 | Κώστας     | Δημητρίου    | 3449.90
 Δυτική Ελλάδα      |   2 | θανάσης    | Καραγιάννης  | 1642.70
 Ήπειρος            |   1 | Ιωάννα     | Παππά        | 2101.80
 Θεσσαλία           |   1 | Σοφία      | Αλεξίου      | 7946.20
 Θεσσαλία           |   2 | Δήμητρα    | Κωνσταντίνου | 1101.10
 Κεντρική Μακεδονία |   1 | Γιώργος    | Παπαδόπουλος | 6630.00
 Κεντρική Μακεδονία |   2 | Βασίλης    | Αντωνίου     | 5177.50
 Κρήτη              |   1 | Κατερίνα   | Σταύρου      | 3786.20
 Κρήτη              |   2 | Αναστασία  | Γεωργίου     | 2816.50
 Νότιο Αιγαίο       |   1 | Σπύρος     | Μαρκόπουλος  |  662.50
 Πελοπόννησος       |   1 | Ευαγγελία  | Χριστοδούλου | 1158.60
(13 rows)
```

Δύο βήματα. Πρώτα το aggregate ανά πελάτη, μετά η κατάταξη ανά περιφέρεια. Το `id` στο `ORDER BY` του παραθύρου σπάει τις πιθανές ισοβαθμίες.

Άσκηση 23.4 · Μερίδιο κάθε παραγγελίας

Για τις παραγγελίες του πελάτη 13 εμφάνισε `id`, αξία και το ποσοστό της αξίας στο σύνολο των παραγγελιών του πελάτη, με ένα δεκαδικό.

ΛΥΣΗ

```
SELECT order_id, total,
       round(100 * total / sum(total) OVER (), 1) AS pct
FROM (SELECT oi.order_id, sum(oi.quantity * oi.unit_price) AS total
      FROM order_items AS oi
      JOIN orders AS o ON o.id = oi.order_id
      WHERE o.customer_id = 13
      GROUP BY oi.order_id) AS t
ORDER BY order_id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 order_id |  total  | pct
----------+---------+------
       20 |  114.90 |  3.0
       35 |  690.80 | 18.2
       40 | 2527.50 | 66.8
       41 |  289.00 |  7.6
       82 |  164.00 |  4.3
(5 rows)
```

Το `OVER ()` χωρίς partition αρκεί, γιατί το `WHERE` έχει ήδη κρατήσει μόνο έναν πελάτη. Για πολλούς πελάτες θα έγραφες `OVER (PARTITION BY customer_id)`.

Άσκηση 23.5 · Διπλές κριτικές με row_number

Βρες με `row_number` τις κριτικές που δεν είναι η πιο πρόσφατη για το ζεύγος προϊόντος και πελάτη τους. Εμφάνισε `id`, `product_id`, `customer_id` και `created_at`.

ΛΥΣΗ

```
SELECT id, product_id, customer_id, created_at
FROM (SELECT id, product_id, customer_id, created_at,
             row_number() OVER (PARTITION BY product_id, customer_id
                                ORDER BY created_at DESC, id DESC) AS rn
      FROM reviews) AS r
WHERE rn > 1;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id | product_id | customer_id |       created_at
----+------------+-------------+------------------------
 25 |         11 |           2 | 2025-07-18 10:03:00+03
(1 row)
```

Είναι το ίδιο πρόβλημα με την άσκηση 17.3, που το έλυσε με `EXISTS`. Εδώ η λύση γενικεύεται εύκολα. Αν ήθελες να κρατήσεις τις δύο πιο πρόσφατες, θα έγραφες `rn > 2`.

Άσκηση 23.6 · Τεταρτημόρια προϊόντων

Χώρισε τα ενεργά προϊόντα με κατηγορία σε τέσσερις ομάδες κατά τιμή, με την 1 για τα ακριβότερα. Εμφάνισε για κάθε ομάδα το πλήθος, τη μικρότερη και τη μεγαλύτερη τιμή.

ΛΥΣΗ

```
SELECT quartile, count(*) AS products, min(price) AS min_price, max(price) AS max_price
FROM (SELECT price, ntile(4) OVER (ORDER BY price DESC) AS quartile
      FROM products
      WHERE active AND category_id IS NOT NULL) AS q
GROUP BY quartile
ORDER BY quartile;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 quartile | products | min_price | max_price
----------+----------+-----------+-----------
        1 |        7 |    249.00 |   2199.00
        2 |        7 |     64.00 |    229.00
        3 |        7 |     34.90 |     59.00
        4 |        7 |      9.90 |     32.00
(4 rows)
```

Τα 28 προϊόντα χωρίζονται σε τέσσερις ομάδες των επτά. Όταν το πλήθος δεν διαιρείται ακριβώς, οι πρώτες ομάδες παίρνουν μία γραμμή παραπάνω.

Άσκηση 23.7 · Η τρίτη παραγγελία κάθε πελάτη

Για τους πελάτες 1 έως 6 εμφάνισε την τρίτη χρονολογικά παραγγελία τους με `id` και ημερομηνία.

ΛΥΣΗ

```
WITH numbered AS (
  SELECT customer_id, id, created_at,
         row_number() OVER (PARTITION BY customer_id ORDER BY created_at, id) AS n
  FROM orders
  WHERE customer_id BETWEEN 1 AND 6
)
SELECT customer_id, id, created_at::date
FROM numbered
WHERE n = 3
ORDER BY customer_id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 customer_id | id | created_at
-------------+----+------------
           1 | 12 | 2025-03-25
           2 | 23 | 2025-05-15
           3 | 32 | 2025-07-29
           4 | 61 | 2025-11-26
           5 | 51 | 2025-10-27
           6 | 26 | 2025-06-18
(6 rows)
```

Με `DISTINCT ON` θα έπαιρνες μόνο την πρώτη γραμμή κάθε ομάδας. Το `row_number` σου δίνει οποιαδήποτε θέση.

ΚΕΦΑΛΑΙΟ 24

Window functions με frames, LAG και LEAD

Άσκηση 24.1 · Σωρευτικά έσοδα ανά μήνα

Εμφάνισε για κάθε μήνα του 2026 τα έσοδα από παραγγελίες που δεν ακυρώθηκαν ούτε επιστράφηκαν και το σωρευτικό σύνολο από την αρχή του έτους.

ΛΥΣΗ

```
WITH monthly AS (
  SELECT date_trunc('month', o.created_at)::date AS month,
         sum(oi.quantity * oi.unit_price) AS revenue
  FROM orders AS o
  JOIN order_items AS oi ON oi.order_id = o.id
  WHERE o.status NOT IN ('cancelled', 'refunded')
    AND o.created_at >= '2026-01-01'
  GROUP BY 1
)
SELECT month, revenue, sum(revenue) OVER (ORDER BY month) AS ytd
FROM monthly
ORDER BY month;
```

ΑΠΟΤΕΛΕΣΜΑ

```
   month    | revenue  |   ytd
------------+----------+----------
 2026-01-01 |  1860.80 |  1860.80
 2026-02-01 |  3747.20 |  5608.00
 2026-03-01 |  1599.20 |  7207.20
 2026-04-01 |  4026.50 | 11233.70
 2026-05-01 | 10338.80 | 21572.50
 2026-06-01 |  9074.90 | 30647.40
(6 rows)
```

Κάθε μήνας εμφανίζεται μία φορά, οπότε δεν υπάρχουν ισοβαθμίες και η προεπιλογή `RANGE` δίνει το ίδιο με το `ROWS`.

Άσκηση 24.2 · Διαστήματα ανάμεσα σε αγορές

Για τον πελάτη 1 εμφάνισε κάθε παραγγελία με ημερομηνία και τις ημέρες από την προηγούμενη. Στην πρώτη παραγγελία δείξε 0 αντί για ∅.

ΛΥΣΗ

```
SELECT id, created_at::date AS day,
       created_at::date - lag(created_at::date, 1, created_at::date)
                          OVER (ORDER BY created_at) AS days_since_previous
FROM orders
WHERE customer_id = 1
ORDER BY created_at;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id  |    day     | days_since_previous
-----+------------+---------------------
  10 | 2025-03-18 |                   0
  11 | 2025-03-19 |                   1
  12 | 2025-03-25 |                   6
  17 | 2025-04-24 |                  30
  21 | 2025-05-08 |                  14
  29 | 2025-07-13 |                  66
  57 | 2025-11-19 |                 129
  58 | 2025-11-21 |                   2
  76 | 2026-01-10 |                  50
  91 | 2026-02-23 |                  44
 122 | 2026-05-18 |                  84
 129 | 2026-05-26 |                   8
(12 rows)
```

Το τρίτο όρισμα του `lag` είναι η τιμή όταν δεν υπάρχει προηγούμενη γραμμή. Εδώ είναι η ίδια ημερομηνία, οπότε η διαφορά βγαίνει 0.

Άσκηση 24.3 · Μεταβολή από μήνα σε μήνα

Ξαναγράψε τη μεταβολή των εσόδων από μήνα σε μήνα της θεωρίας με ένα CTE για τα μηνιαία έσοδα. Εμφάνισε μήνα, έσοδα, έσοδα του προηγούμενου μήνα και ποσοστό μεταβολής με ένα δεκαδικό.

ΛΥΣΗ

```
WITH monthly AS (
  SELECT date_trunc('month', o.created_at)::date AS month,
         sum(oi.quantity * oi.unit_price) AS revenue
  FROM orders AS o
  JOIN order_items AS oi ON oi.order_id = o.id
  WHERE o.status NOT IN ('cancelled', 'refunded')
    AND o.created_at >= '2026-01-01'
  GROUP BY 1
)
SELECT month, revenue,
       lag(revenue) OVER (ORDER BY month) AS previous,
       round(100 * (revenue / lag(revenue) OVER (ORDER BY month) - 1), 1) AS change_pct
FROM monthly
ORDER BY month;
```

ΑΠΟΤΕΛΕΣΜΑ

```
   month    | revenue  | previous | change_pct
------------+----------+----------+------------
 2026-01-01 |  1860.80 |        ∅ |          ∅
 2026-02-01 |  3747.20 |  1860.80 |      101.4
 2026-03-01 |  1599.20 |  3747.20 |      -57.3
 2026-04-01 |  4026.50 |  1599.20 |      151.8
 2026-05-01 | 10338.80 |  4026.50 |      156.8
 2026-06-01 |  9074.90 | 10338.80 |      -12.2
(6 rows)
```

Το CTE κάνει το aggregate και το κύριο query τη σύγκριση. Κάθε έκφραση έχει πια ένα μόνο επίπεδο.

Άσκηση 24.4 · Εβδομαδιαίος κινούμενος μέσος

Για τις ημέρες του Ιουνίου 2026 με παραγγελίες, εμφάνισε τα έσοδα και τον μέσο όρο των εσόδων των ημερών με παραγγελίες μέσα στις επτά τελευταίες ημερολογιακές ημέρες, μαζί με την τρέχουσα. Κράτα τις γραμμές από τις 20 Ιουνίου και μετά.

ΛΥΣΗ

```
WITH daily AS (
  SELECT o.created_at::date AS day, sum(oi.quantity * oi.unit_price) AS revenue
  FROM orders AS o
  JOIN order_items AS oi ON oi.order_id = o.id
  WHERE o.status NOT IN ('cancelled', 'refunded')
    AND o.created_at >= '2026-06-01'
  GROUP BY 1
),
smoothed AS (
  SELECT day, revenue,
         round(avg(revenue) OVER (ORDER BY day RANGE BETWEEN interval '6 days' PRECEDING
                                  AND CURRENT ROW), 2) AS avg_7d
  FROM daily
)
SELECT * FROM smoothed WHERE day >= '2026-06-20' ORDER BY day;
```

ΑΠΟΤΕΛΕΣΜΑ

```
    day     | revenue | avg_7d
------------+---------+--------
 2026-06-21 |  688.80 | 373.48
 2026-06-23 |  751.40 | 428.68
 2026-06-24 | 1316.00 | 700.68
 2026-06-25 |  109.80 | 582.50
 2026-06-26 |  133.50 | 599.90
(5 rows)
```

Το φίλτρο των ημερών μπαίνει σε ξεχωριστό βήμα. Αν έμπαινε στο `daily`, οι ημέρες πριν από τις 20 Ιουνίου δεν θα υπήρχαν για να μπουν στο frame των πρώτων γραμμών.

Άσκηση 24.5 · Πρώτη και τελευταία αγορά

Για τις παραγγελίες του πελάτη 6 εμφάνισε `id`, ημερομηνία, την ημερομηνία της πρώτης και της τελευταίας παραγγελίας του πελάτη, χωρίς `last_value`.

ΛΥΣΗ

```
SELECT id, created_at::date AS day,
       first_value(created_at::date) OVER (ORDER BY created_at) AS first_day,
       first_value(created_at::date) OVER (ORDER BY created_at DESC) AS last_day
FROM orders
WHERE customer_id = 6
ORDER BY created_at;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id  |    day     | first_day  |  last_day
-----+------------+------------+------------
   2 | 2025-01-19 | 2025-01-19 | 2026-06-13
  25 | 2025-05-30 | 2025-01-19 | 2026-06-13
  26 | 2025-06-18 | 2025-01-19 | 2026-06-13
  39 | 2025-09-16 | 2025-01-19 | 2026-06-13
  42 | 2025-09-25 | 2025-01-19 | 2026-06-13
  45 | 2025-10-01 | 2025-01-19 | 2026-06-13
  55 | 2025-11-10 | 2025-01-19 | 2026-06-13
 102 | 2026-03-25 | 2025-01-19 | 2026-06-13
 121 | 2026-05-17 | 2025-01-19 | 2026-06-13
 143 | 2026-06-13 | 2025-01-19 | 2026-06-13
(10 rows)
```

Το `first_value` με φθίνουσα σειρά βρίσκει την τελευταία ημερομηνία. Το frame ξεκινά πάντα από την πρώτη γραμμή του partition, οπότε δεν χρειάζεται να το αλλάξεις. Ένα query μπορεί να έχει window functions με διαφορετική σειρά.

Άσκηση 24.6 · Νέοι και επαναλαμβανόμενοι πελάτες

Για κάθε μήνα του 2026 μέτρησε πόσες παραγγελίες ήταν η πρώτη του πελάτη τους και πόσες όχι. Λάβε υπόψη όλο το ιστορικό, όχι μόνο το 2026.

ΛΥΣΗ

```
WITH numbered AS (
  SELECT id, created_at,
         row_number() OVER (PARTITION BY customer_id ORDER BY created_at, id) AS n
  FROM orders
)
SELECT date_trunc('month', created_at)::date AS month,
       count(*) FILTER (WHERE n = 1) AS first_orders,
       count(*) FILTER (WHERE n > 1) AS repeat_orders
FROM numbered
WHERE created_at >= '2026-01-01'
GROUP BY 1
ORDER BY 1;
```

ΑΠΟΤΕΛΕΣΜΑ

```
   month    | first_orders | repeat_orders
------------+--------------+---------------
 2026-01-01 |            2 |             4
 2026-02-01 |            1 |            11
 2026-03-01 |            2 |            10
 2026-04-01 |            1 |             8
 2026-05-01 |            0 |            20
 2026-06-01 |            0 |            29
(6 rows)
```

Η αρίθμηση γίνεται πάνω σε όλες τις παραγγελίες και το φίλτρο του 2026 στο εξωτερικό query. Αν το `WHERE` ήταν μέσα στο CTE, η πρώτη παραγγελία του 2026 κάθε παλιού πελάτη θα φαινόταν ως πρώτη αγορά.

ΚΕΦΑΛΑΙΟ 25

Gaps and islands και χρονοσειρές

Άσκηση 25.1 · Παραγγελίες κάθε ημέρας

Εμφάνισε για κάθε ημέρα από 20 έως 31 Μαΐου 2026 το πλήθος των παραγγελιών, οποιασδήποτε κατάστασης, με 0 για τις ημέρες χωρίς παραγγελίες.

ΛΥΣΗ

```
SELECT d::date AS day, count(o.id) AS orders
FROM generate_series(date '2026-05-20', date '2026-05-31', interval '1 day') AS d
LEFT JOIN orders AS o ON o.created_at::date = d::date
GROUP BY d
ORDER BY d;
```

ΑΠΟΤΕΛΕΣΜΑ

```
    day     | orders
------------+--------
 2026-05-20 |      0
 2026-05-21 |      1
 2026-05-22 |      0
 2026-05-23 |      1
 2026-05-24 |      1
 2026-05-25 |      2
 2026-05-26 |      2
 2026-05-27 |      0
 2026-05-28 |      0
 2026-05-29 |      1
 2026-05-30 |      1
 2026-05-31 |      1
(12 rows)
```

Το `count(o.id)` μετρά μόνο τις γραμμές όπου βρέθηκε παραγγελία, οπότε οι κενές ημέρες βγαίνουν 0. Το `count(*)` θα έδινε 1 για αυτές. Το σύντομο αυτό query κάνει το join και την ομαδοποίηση σε ένα βήμα, χωρίς CTE.

Άσκηση 25.2 · Εβδομάδες του 2026

Εμφάνισε τα έσοδα κάθε εβδομάδας του πρώτου τριμήνου του 2026, ξεκινώντας από τη Δευτέρα 29 Δεκεμβρίου 2025, από παραγγελίες που δεν ακυρώθηκαν ούτε επιστράφηκαν. Οι εβδομάδες χωρίς έσοδα πρέπει να έχουν 0.

ΛΥΣΗ

```
WITH weeks AS (
  SELECT w::date AS week
  FROM generate_series(date '2025-12-29', date '2026-03-30', interval '7 days') AS w
),
weekly AS (
  SELECT date_trunc('week', o.created_at)::date AS week,
         sum(oi.quantity * oi.unit_price) AS revenue
  FROM orders AS o
  JOIN order_items AS oi ON oi.order_id = o.id
  WHERE o.status NOT IN ('cancelled', 'refunded')
  GROUP BY 1
)
SELECT weeks.week, COALESCE(weekly.revenue, 0) AS revenue
FROM weeks
LEFT JOIN weekly ON weekly.week = weeks.week
ORDER BY weeks.week;
```

ΑΠΟΤΕΛΕΣΜΑ

```
    week    | revenue
------------+---------
 2025-12-29 |  324.90
 2026-01-05 |   79.90
 2026-01-12 |       0
 2026-01-19 | 1149.00
 2026-01-26 |  528.00
 2026-02-02 |  416.60
 2026-02-09 | 1584.00
 2026-02-16 |   45.60
 2026-02-23 | 1829.90
 2026-03-02 |  844.40
 2026-03-09 |       0
 2026-03-16 |  248.00
 2026-03-23 |  368.00
 2026-03-30 |   54.60
(14 rows)
```

Το `date_trunc('week', ...)` δίνει τη Δευτέρα κάθε εβδομάδας. Γι' αυτό το ημερολόγιο ξεκινά από Δευτέρα, ώστε τα κλειδιά των δύο πλευρών να ταιριάζουν.

Άσκηση 25.3 · Κενά του Μαΐου

Βρες τα διαστήματα του Μαΐου 2026 χωρίς καμία παραγγελία, με αρχή, τέλος και πλήθος ημερών. Λάβε υπόψη μόνο παραγγελίες του Μαΐου.

ΛΥΣΗ

```
WITH order_days AS (
  SELECT DISTINCT created_at::date AS day
  FROM orders
  WHERE created_at >= '2026-05-01' AND created_at < '2026-06-01'
)
SELECT day + 1 AS gap_start, next_day - 1 AS gap_end, next_day - day - 1 AS empty_days
FROM (SELECT day, lead(day) OVER (ORDER BY day) AS next_day FROM order_days) AS t
WHERE next_day - day > 1
ORDER BY gap_start;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 gap_start  |  gap_end   | empty_days
------------+------------+------------
 2026-05-06 | 2026-05-06 |          1
 2026-05-08 | 2026-05-08 |          1
 2026-05-10 | 2026-05-10 |          1
 2026-05-13 | 2026-05-16 |          4
 2026-05-19 | 2026-05-20 |          2
 2026-05-22 | 2026-05-22 |          1
 2026-05-27 | 2026-05-28 |          2
(7 rows)
```

Η λύση δεν βλέπει κενά πριν από την πρώτη ή μετά την τελευταία ημέρα με παραγγελίες του μήνα, αφού εκεί δεν υπάρχει ζεύγος διαδοχικών ημερών. Για αυτά θα χρειαζόσουν το ημερολόγιο με anti join.

Άσκηση 25.4 · Το μεγαλύτερο σερί

Βρες το μεγαλύτερο συνεχόμενο διάστημα ημερών του 2026 με τουλάχιστον μία παραγγελία. Εμφάνισε αρχή, τέλος και πλήθος ημερών. Αν υπάρχουν ισοβαθμίες, δείξε όλα τα διαστήματα με το μέγιστο μήκος.

ΛΥΣΗ

```
WITH order_days AS (
  SELECT DISTINCT created_at::date AS day
  FROM orders
  WHERE created_at >= '2026-01-01'
),
islands AS (
  SELECT min(day) AS island_start, max(day) AS island_end, count(*) AS days
  FROM (SELECT day, day - row_number() OVER (ORDER BY day)::integer AS island
        FROM order_days) AS m
  GROUP BY island
)
SELECT * FROM islands
ORDER BY days DESC
FETCH FIRST 1 ROWS WITH TIES;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 island_start | island_end | days
--------------+------------+------
 2026-05-23   | 2026-05-26 |    4
 2026-06-03   | 2026-06-06 |    4
 2026-06-23   | 2026-06-26 |    4
(3 rows)
```

Το `DISTINCT` στο πρώτο CTE είναι απαραίτητο. Αν μια ημέρα εμφανιζόταν δύο φορές, το `row_number` θα αυξανόταν ενώ η ημερομηνία όχι. Το island θα έσπαγε στα δύο.

Υπάρχουν τρία σερί τεσσάρων ημερών και το `WITH TIES` τα δείχνει όλα. Με `LIMIT 1` θα έβλεπες μόνο ένα, χωρίς να ξέρεις ότι ισοβαθμεί με άλλα.

Άσκηση 25.5 · Οι συνεδρίες του πελάτη 3

Για τον πελάτη 3 μέτρησε τις συνεδρίες με τον κανόνα των 30 λεπτών. Εμφάνισε το πλήθος των συνεδριών, τον μέσο αριθμό events ανά συνεδρία με ένα δεκαδικό και πόσες συνεδρίες περιέχουν `checkout`.

ΛΥΣΗ

```
WITH flagged AS (
  SELECT occurred_at, kind,
         CASE WHEN occurred_at - lag(occurred_at) OVER (ORDER BY occurred_at)
                   <= interval '30 minutes' THEN 0 ELSE 1 END AS new_session
  FROM events
  WHERE customer_id = 3
),
numbered AS (
  SELECT *, sum(new_session) OVER (ORDER BY occurred_at ROWS UNBOUNDED PRECEDING) AS session_no
  FROM flagged
),
sessions AS (
  SELECT session_no, count(*) AS events, bool_or(kind = 'checkout') AS converted
  FROM numbered
  GROUP BY session_no
)
SELECT count(*) AS sessions,
       round(avg(events), 1) AS avg_events,
       count(*) FILTER (WHERE converted) AS with_checkout
FROM sessions;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 sessions | avg_events | with_checkout
----------+------------+---------------
       18 |        4.2 |             8
(1 row)
```

Ένας πελάτης αρκεί, οπότε το `PARTITION BY customer_id` της θεωρίας δεν χρειάζεται. Το `bool_or` λέει αν τουλάχιστον ένα event της συνεδρίας ήταν `checkout`.

Άσκηση 25.6 · Πελάτες που χάθηκαν για καιρό

Βρες τα διαστήματα πάνω από 120 ημέρες ανάμεσα σε δύο διαδοχικές παραγγελίες του ίδιου πελάτη. Εμφάνισε πελάτη, ημερομηνία της πρώτης, ημερομηνία της επόμενης και ημέρες, από το μεγαλύτερο διάστημα.

ΛΥΣΗ

```
SELECT customer_id, day AS from_day, next_day AS to_day, next_day - day AS days
FROM (SELECT customer_id, created_at::date AS day,
             lead(created_at::date) OVER (PARTITION BY customer_id ORDER BY created_at) AS next_day
      FROM orders) AS t
WHERE next_day - day > 120
ORDER BY days DESC, customer_id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 customer_id |  from_day  |   to_day   | days
-------------+------------+------------+------
           5 | 2025-02-07 | 2025-10-27 |  262
           3 | 2025-09-04 | 2026-04-23 |  231
          18 | 2025-11-05 | 2026-06-08 |  215
           4 | 2025-05-02 | 2025-11-26 |  208
           7 | 2025-10-17 | 2026-05-09 |  204
           9 | 2025-04-03 | 2025-10-19 |  199
           8 | 2025-03-28 | 2025-09-30 |  186
          17 | 2025-12-19 | 2026-06-08 |  171
          13 | 2025-09-21 | 2026-02-04 |  136
           6 | 2025-11-10 | 2026-03-25 |  135
           6 | 2025-01-19 | 2025-05-30 |  131
           1 | 2025-07-13 | 2025-11-19 |  129
          12 | 2025-07-18 | 2025-11-19 |  124
          14 | 2025-12-08 | 2026-04-10 |  123
 …
(15 rows)
```

Το `PARTITION BY customer_id` κρατά το `lead` μέσα στις παραγγελίες του ίδιου πελάτη. Χωρίς αυτό, η τελευταία παραγγελία ενός πελάτη θα συγκρινόταν με την πρώτη του επόμενου.

ΚΕΦΑΛΑΙΟ 26

GROUPING SETS, ROLLUP, CUBE και pivot

Άσκηση 26.1 · Υπάλληλοι ανά τμήμα και θέση

Εμφάνισε το πλήθος και το άθροισμα μισθών των υπαλλήλων με τμήμα, ανά τμήμα και θέση, με υποσύνολα ανά τμήμα και γενικό σύνολο. Χρησιμοποίησε `ROLLUP`.

ΛΥΣΗ

```
SELECT department, title, count(*) AS employees, sum(salary) AS payroll
FROM employees
WHERE department IS NOT NULL
GROUP BY ROLLUP (department, title)
ORDER BY department, title;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 department |       title        | employees | payroll
------------+--------------------+-----------+----------
 Αποθήκη    | Αποθηκάριος        |         2 |  3550.00
 Αποθήκη    | Οδηγός διανομής    |         1 |  1650.00
 Αποθήκη    | Operations Manager |         1 |  6200.00
 Αποθήκη    | ∅                  |         4 | 11400.00
 Διοίκηση   | CEO                |         1 |  9500.00
 Διοίκηση   | ∅                  |         1 |  9500.00
 Πωλήσεις   | Account Manager    |         2 |  6000.00
 Πωλήσεις   | Head of Sales      |         1 |  7000.00
 Πωλήσεις   | Sales Intern       |         1 |  1100.00
 Πωλήσεις   | ∅                  |         4 | 14100.00
 Τεχνολογία | Backend Developer  |         1 |  4200.00
 Τεχνολογία | CTO                |         1 |  8200.00
 Τεχνολογία | Data Engineer      |         1 |  4400.00
 Τεχνολογία | Frontend Developer |         1 |  3900.00
 …
(17 rows)
```

Οι γραμμές με ∅ στη θέση είναι τα υποσύνολα των τμημάτων. Η τελευταία, με ∅ και στα δύο, είναι το γενικό σύνολο. Η ταξινόμηση βάζει το ∅ στο τέλος κάθε ομάδας, επειδή το `NULL` θεωρείται μεγαλύτερο από κάθε τιμή.

Άσκηση 26.2 · Ετικέτες συνόλων

Ξαναγράψε την προηγούμενη άσκηση ώστε οι γραμμές συνόλων να γράφουν `όλα τα τμήματα` και `όλες οι θέσεις` αντί για ∅. Κράτα μόνο τις στήλες τμήματος, θέσης και πλήθους.

ΛΥΣΗ

```
SELECT CASE WHEN GROUPING(department) = 1 THEN 'όλα τα τμήματα' ELSE department END AS department,
       CASE WHEN GROUPING(title) = 1 THEN 'όλες οι θέσεις' ELSE title END AS title,
       count(*) AS employees
FROM employees
WHERE department IS NOT NULL
GROUP BY ROLLUP (department, title)
ORDER BY GROUPING(department), 1, GROUPING(title), 2;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 department |       title        | employees
------------+--------------------+-----------
 Αποθήκη    | Αποθηκάριος        |         2
 Αποθήκη    | Οδηγός διανομής    |         1
 Αποθήκη    | Operations Manager |         1
 Αποθήκη    | όλες οι θέσεις     |         4
 Διοίκηση   | CEO                |         1
 Διοίκηση   | όλες οι θέσεις     |         1
 Πωλήσεις   | Account Manager    |         2
 Πωλήσεις   | Head of Sales      |         1
 Πωλήσεις   | Sales Intern       |         1
 Πωλήσεις   | όλες οι θέσεις     |         4
 Τεχνολογία | Backend Developer  |         1
 Τεχνολογία | CTO                |         1
 Τεχνολογία | Data Engineer      |         1
 Τεχνολογία | Frontend Developer |         1
 …
(17 rows)
```

Το `GROUPING` στο `ORDER BY` κρατά τις γραμμές συνόλων στο τέλος της ομάδας τους, ανεξάρτητα από το πώς ταξινομείται το κείμενο της ετικέτας.

Άσκηση 26.3 · Κύβος πληρωμών

Με `CUBE`, υπολόγισε το πλήθος των θετικών πληρωμών ανά μέθοδο και ανά τρίμηνο του 2026, με όλους τους συνδυασμούς συνόλων. Το τρίμηνο να είναι αριθμός από 1 έως 4.

ΛΥΣΗ

```
SELECT method, extract(quarter FROM paid_at) AS quarter, count(*) AS payments
FROM payments
WHERE amount > 0 AND paid_at >= '2026-01-01'
GROUP BY CUBE (method, extract(quarter FROM paid_at))
ORDER BY method, quarter;
```

ΑΠΟΤΕΛΕΣΜΑ

```
  method  | quarter | payments
----------+---------+----------
 bank     |       1 |        1
 bank     |       2 |        6
 bank     |       ø |        7
 card     |       1 |       19
 card     |       2 |       31
 card     |       ø |       50
 cod      |       1 |        4
 cod      |       2 |        7
 cod      |       ø |       11
 giftcard |       1 |        3
 giftcard |       2 |        3
 giftcard |       ø |        6
 paypal   |       1 |        4
 paypal   |       2 |        8
 …
(18 rows)
```

Το `extract(quarter ...)` είναι ένα ακόμη πεδίο της συνάρτησης του κεφαλαίου 6. Η έκφραση του `GROUP BY` πρέπει να είναι ίδια με αυτήν του `SELECT`.

Άσκηση 26.4 · Πίνακας μηνών και μεθόδων

Φτιάξε pivot με μία γραμμή ανά μήνα του 2026 και μία στήλη για το καθαρό ποσό κάθε μεθόδου πληρωμής `card`, `paypal`, `bank` και `cod`. Όπου δεν υπάρχει ποσό γράψε 0.

ΛΥΣΗ

```
SELECT to_char(paid_at, 'YYYY-MM') AS month,
       COALESCE(sum(amount) FILTER (WHERE method = 'card'), 0) AS card,
       COALESCE(sum(amount) FILTER (WHERE method = 'paypal'), 0) AS paypal,
       COALESCE(sum(amount) FILTER (WHERE method = 'bank'), 0) AS bank,
       COALESCE(sum(amount) FILTER (WHERE method = 'cod'), 0) AS cod
FROM payments
WHERE paid_at >= '2026-01-01'
GROUP BY 1
ORDER BY 1;
```

ΑΠΟΤΕΛΕΣΜΑ

```
  month  |  card   | paypal  |  bank   |   cod
---------+---------+---------+---------+---------
 2026-01 |  278.90 | 1149.00 |       0 |  103.90
 2026-02 | 2357.70 |  128.00 |       0 | 1540.50
 2026-03 | 1366.30 |   30.40 |   20.50 |   82.00
 2026-04 | 1043.70 | 2208.90 |  773.90 |       0
 2026-05 | 6722.70 |  649.00 | 2400.10 |  567.00
 2026-06 | 4462.70 | 1013.60 |  817.00 | 1081.70
(6 rows)
```

Το `sum` με `FILTER` χωρίς γραμμές δίνει `NULL`, οπότε το `COALESCE` είναι απαραίτητο για να εμφανίζεται 0. Οι επιστροφές μειώνουν σωστά το ποσό της μεθόδου τους.

Άσκηση 26.5 · Διάμεσος ανά κατηγορία

Για κάθε κατηγορία με ενεργά προϊόντα εμφάνισε τη μέση και τη διάμεσο τιμή τους, με δύο δεκαδικά. Εμφάνισε μόνο τις κατηγορίες όπου ο μέσος όρος ξεπερνά τη διάμεσο κατά περισσότερο από 10%.

ΛΥΣΗ

```
SELECT cat.name AS category,
       round(avg(p.price), 2) AS mean,
       round(percentile_cont(0.5) WITHIN GROUP (ORDER BY p.price)::numeric, 2) AS median
FROM products AS p
JOIN categories AS cat ON cat.id = p.category_id
WHERE p.active
GROUP BY cat.name
HAVING avg(p.price) > 1.1 * percentile_cont(0.5) WITHIN GROUP (ORDER BY p.price)
ORDER BY cat.name;
```

ΑΠΟΤΕΛΕΣΜΑ

```
   category    |  mean   | median
---------------+---------+---------
 Κουζίνα       |  115.97 |   64.00
 Περιφερειακά  |  128.18 |   89.00
 Laptops       | 1332.33 | 1149.00
(3 rows)
```

Όταν ο μέσος όρος είναι πολύ πάνω από τη διάμεσο, λίγα ακριβά προϊόντα τραβούν τον μέσο όρο προς τα πάνω. Στα laptops αυτό είναι το `Laptop Pro 16`.

Άσκηση 26.6 · Η συνηθισμένη ώρα

Για κάθε ημέρα της εβδομάδας, με 1 τη Δευτέρα, βρες την ώρα της ημέρας με τις περισσότερες παραγγελίες, με `mode()`.

ΛΥΣΗ

```
SELECT extract(isodow FROM created_at) AS weekday,
       mode() WITHIN GROUP (ORDER BY extract(hour FROM created_at)) AS busiest_hour,
       count(*) AS orders
FROM orders
GROUP BY 1
ORDER BY 1;
```

```
ΑΠΟΤΕΛΕΣΜΑ

 weekday | busiest_hour | orders
---------+--------------+--------
       1 |           14 |     31
       2 |            9 |     23
       3 |           11 |     21
       4 |            8 |     22
       5 |           13 |     22
       6 |           22 |     15
       7 |           22 |     28
(7 rows)
```

Όταν δύο ώρες ισοβαθμούν, το `mode()` επιστρέφει τη μικρότερη, αφού διαβάζει τις τιμές με τη σειρά του `WITHIN GROUP`.

ΚΕΦΑΛΑΙΟ 27

Arrays και JSONB

Άσκηση 27.1 · Προϊόντα γραφείου

Βρες τα προϊόντα που έχουν την ετικέτα `γραφείο` ή την ετικέτα `usb-c`. Εμφάνισε όνομα και ετικέτες.

ΛΥΣΗ

```
SELECT name, tags
FROM products
WHERE tags && ARRAY['γραφείο', 'usb-c']
ORDER BY name;
```

ΑΠΟΤΕΛΕΣΜΑ

```
            name             |         tags
-----------------------------+----------------------
 Γραφείο ρυθμιζόμενου ύψους | {γραφείο,εργονομία}
 Εργονομική καρέκλα         | {γραφείο,εργονομία}
 Φωτιστικό γραφείου LED     | {γραφείο}
 USB-C docking station      | {περιφερειακά,usb-c}
(4 rows)
```

Το `&&` ζητά τουλάχιστον ένα κοινό στοιχείο. Με `@>` θα ζητούσες και τα δύο.

Άσκηση 27.2 · Πωλήσεις ανά ετικέτα

Υπολόγισε τα τεμάχια που πουλήθηκαν για κάθε ετικέτα προϊόντος σε παραγγελίες που δεν ακυρώθηκαν ούτε επιστράφηκαν. Εμφάνισε τις πέντε ετικέτες με τα περισσότερα τεμάχια.

ΛΥΣΗ

```
SELECT tag, sum(oi.quantity) AS units
FROM order_items AS oi
JOIN orders AS o ON o.id = oi.order_id
JOIN products AS p ON p.id = oi.product_id
CROSS JOIN LATERAL unnest(p.tags) AS tag
WHERE o.status NOT IN ('cancelled', 'refunded')
GROUP BY tag
ORDER BY units DESC, tag
LIMIT 5;
```

ΑΠΟΤΕΛΕΣΜΑ

```
       tag       | units
-----------------+-------
 βιβλίο          |   135
 λογοτεχνία      |    56
 περιφερειακά    |    48
 προγραμματισμός |    39
 γραφείο         |    36
(5 rows)
```

Ένα προϊόν με τρεις ετικέτες μετρά σε τρεις ομάδες. Αυτό είναι σωστό εδώ, αφού ρωτάμε ανά ετικέτα. Το άθροισμα όλων των γραμμών δεν είναι όμως τα συνολικά τεμάχια, γιατί κάθε πώληση μετρά μία φορά για κάθε ετικέτα.

Άσκηση 27.3 · Laptops με μνήμη

Εμφάνισε όνομα, μνήμη και αποθηκευτικό χώρο των laptops με τουλάχιστον 16 GB μνήμη, ως ακέραιους, ταξινομημένα από τη μεγαλύτερη μνήμη.

ΛΥΣΗ

```
SELECT name,
       (attrs ->> 'ram_gb')::integer AS ram_gb,
       (attrs ->> 'storage_gb')::integer AS storage_gb
FROM products
WHERE category_id = 8
  AND (attrs ->> 'ram_gb')::integer >= 16
ORDER BY ram_gb DESC;
```

ΑΠΟΤΕΛΕΣΜΑ

```
      name      | ram_gb | storage_gb
----------------+--------+------------
 Laptop Pro 16  |     32 |       1024
 Laptop Aero 14 |     16 |        512
(2 rows)
```

Χωρίς τη μετατροπή σε `integer`, η σύγκριση θα ήταν ανάμεσα σε κείμενα και το `'8' >= '16'` θα ήταν αληθές, αφού το 8 έρχεται μετά το 1 αλφαβητικά.

Άσκηση 27.4 · Οι συχνότερες αναζητήσεις

Από τα events τύπου `search`, βρες τις πέντε συχνότερες αναζητήσεις με το πλήθος τους και πόσοι διαφορετικοί πελάτες τις έκαναν.

ΛΥΣΗ

```
SELECT payload ->> 'q' AS query, count(*) AS searches,
       count(DISTINCT customer_id) AS customers
FROM events
WHERE kind = 'search'
GROUP BY payload ->> 'q'
ORDER BY searches DESC, query
LIMIT 5;
```

ΑΠΟΤΕΛΕΣΜΑ

```
      query       | searches | customers
------------------+----------+-----------
 postgres         |       26 |        18
 ακουστικά ANC    |       24 |        14
 καρεκλα γραφειου |       23 |        15
 καφετιερα        |       22 |        12
 python           |       20 |        19
(5 rows)
```

Οι αναζητήσεις `καρεκλα γραφειου` και `καφετιερα` είναι χωρίς τόνους. Το επόμενο κεφάλαιο δείχνει πώς τις αντιστοιχίζεις σε προϊόντα παρά τα λάθη.

Άσκηση 27.5 · Πελάτης ως JSON

Φτιάξε για τον πελάτη 13 ένα έγγραφο JSON με το ονοματεπώνυμο, την πόλη και ένα array με τις παραγγελίες του, όπου κάθε παραγγελία έχει `id` και ημερομηνία ως κείμενο `YYYY-MM-DD`. Εμφάνισέ το με `jsonb_pretty`.

ΛΥΣΗ

```
SELECT jsonb_pretty(jsonb_build_object(
         'name', c.first_name || ' ' || c.last_name,
         'city', t.name,
         'orders', jsonb_agg(jsonb_build_object(
                     'id', o.id,
                     'date', to_char(o.created_at, 'YYYY-MM-DD'))
                   ORDER BY o.created_at))) AS doc
FROM customers AS c
JOIN cities AS t ON t.id = c.city_id
JOIN orders AS o ON o.customer_id = c.id
WHERE c.id = 13
GROUP BY c.id, t.name;
```

ΑΠΟΤΕΛΕΣΜΑ

```
               doc
---------------------------------
 {                               +
     "city": "Χανιά",            +
     "name": "Κατερίνα Σταύρου", +
     "orders": [                 +
         {                       +
             "id": 20,           +
             "date": "2025-05-04"+
         },                      +
         {                       +
             "id": 35,           +
             "date": "2025-09-03"+
         },                      +
         {                       +
             "id": 40,           +
             "date": "2025-09-19"+
         },                      +
         {                       +
             "id": 41,           +
             "date": "2025-09-21"+
         },                      +
         {                       +
             "id": 82,           +
             "date": "2026-02-04"+
         }                       +
     ]                           +
 }
(1 row)
```

Όλο το έγγραφο φτιάχνεται στη βάση με ένα query. Η εφαρμογή το στέλνει όπως είναι, χωρίς να συναρμολογεί JSON από πολλά queries.

Άσκηση 27.6 · Θύρες σε γραμμές

Με `JSON_TABLE`, εμφάνισε για κάθε προϊόν με κλειδί `ports` το όνομά του και κάθε θύρα σε δική της γραμμή, με τη θέση της στη λίστα.

ΛΥΣΗ

```
SELECT p.name, pt.pos, pt.port
FROM products AS p
CROSS JOIN LATERAL JSON_TABLE(p.attrs, '$.ports[*]'
                              COLUMNS (pos FOR ORDINALITY,
                                       port text PATH '$')) AS pt
WHERE p.attrs ? 'ports'
ORDER BY p.name, pt.pos;
```

ΑΠΟΤΕΛΕΣΜΑ

```
         name          | pos |   port
-----------------------+-----+----------
 Οθόνη 27" 4K          |   1 | hdmi
 Οθόνη 27" 4K          |   2 | usb-c
 USB-C docking station |   1 | hdmi
 USB-C docking station |   2 | ethernet
 USB-C docking station |   3 | usb-a
 USB-C docking station |   4 | usb-c
(6 rows)
```

Το `FOR ORDINALITY` στο `JSON_TABLE` κάνει ό,τι το `WITH ORDINALITY` στο `unnest`. Αριθμεί τις γραμμές από το 1.

Άσκηση 27.7 · Εγγύηση και αφαίρεση κλειδιού

Σε ένα query, πρόσθεσε στα προϊόντα της κατηγορίας 10 το κλειδί `warranty_years` με τιμή 1 και αφαίρεσε από όλα τα προϊόντα το κλειδί `colors`. Εμφάνισε για τις γραμμές της κατηγορίας 10 που άλλαξαν το όνομα, την εγγύηση και αν έχουν ακόμη κλειδί `colors`.

ΛΥΣΗ

```
WITH changed AS (
  UPDATE products
  SET attrs = (attrs - 'colors') ||
              CASE WHEN category_id = 10 THEN '{"warranty_years": 1}'::jsonb
                   ELSE '{}'::jsonb END
  WHERE attrs ? 'colors' OR category_id = 10
  RETURNING name, category_id, attrs
)
SELECT name, attrs ->> 'warranty_years' AS warranty, attrs ? 'colors' AS has_colors
FROM changed
WHERE category_id = 10
ORDER BY name;
```

ΑΠΟΤΕΛΕΣΜΑ

```
      name       | warranty | has_colors
-----------------+----------+------------
 Ακουστικά ANC   | 1        | f
 Ηχείο Bluetooth | 1        | f
 Μικρόφωνο USB   | 1        | f
 Πικάπ           | 1        | f
(4 rows)
```

Ένα `UPDATE` κάνει και τις δύο αλλαγές. Το `WHERE` του `UPDATE` περιορίζει τις γραμμές σε όσες αλλάζουν πραγματικά. Το CTE του κεφαλαίου 21 επιτρέπει να δείξεις μόνο ένα μέρος των γραμμών που άλλαξαν, χωρίς να επηρεάσει ποιες άλλαξαν.

ΚΕΦΑΛΑΙΟ 28

Αναζήτηση κειμένου με regex, trigrams και full text search

Άσκηση 28.1 · Email με κεφαλαία

Βρες με regular expression τους πελάτες που το email τους περιέχει τουλάχιστον ένα κεφαλαίο λατινικό γράμμα. Εμφάνισε `id` και email.

ΛΥΣΗ

```
SELECT id, email
FROM customers
WHERE email ~ '[A-Z]';
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id |              email
----+----------------------------------
 21 | Giorgos.Papadopoulos@Example.gr
(1 row)
```

Η κλάση `[A-Z]` ταιριάζει με ένα κεφαλαίο λατινικό γράμμα οπουδήποτε στο κείμενο, αφού το μοτίβο δεν έχει `^` ούτε `$`. Το email του πελάτη 21 είναι ο λόγος που το κεφάλαιο 17 συζήτησε μοναδικότητα με `lower(email)`.

Άσκηση 28.2 · Αναζήτηση χωρίς τόνους

Βρες τα προϊόντα που το όνομα ή η περιγραφή τους περιέχει τη λέξη `ηχειο` ή τη λέξη `μικροφωνο`, χωρίς να σε ενδιαφέρουν τόνοι και κεφαλαία.

ΛΥΣΗ

```
SELECT name
FROM products
WHERE unaccent(name || ' ' || description) ~* 'ηχειο|μικροφωνο'
ORDER BY name;
```

ΑΠΟΤΕΛΕΣΜΑ

```
      name
-----------------
 Ηχείο Bluetooth
 Μικρόφωνο USB
(2 rows)
```

Το `~*` κάνει σύγκριση χωρίς διάκριση πεζών και κεφαλαίων και το `|` σημαίνει «ή» μέσα στο regular expression. Το `unaccent` εφαρμόζεται στο ενωμένο κείμενο μία φορά.

Άσκηση 28.3 · Από αναζήτηση σε προϊόν

Για κάθε διαφορετική αναζήτηση των events, βρες το προϊόν με τη μεγαλύτερη `word_similarity` ανάμεσα στην αναζήτηση χωρίς τόνους και σε πεζά και στο όνομα του προϊόντος χωρίς τόνους και σε πεζά. Εμφάνισε μόνο όσες έχουν βαθμό τουλάχιστον 0.5.

ΛΥΣΗ

```
WITH queries AS (
  SELECT DISTINCT lower(unaccent(payload ->> 'q')) AS q
  FROM events
  WHERE kind = 'search'
)
SELECT q.q, best.name, round(best.score::numeric, 2) AS score
FROM queries AS q
CROSS JOIN LATERAL (
  SELECT p.name, word_similarity(q.q, lower(unaccent(p.name))) AS score
  FROM products AS p
  ORDER BY score DESC
  LIMIT 1
) AS best
WHERE best.score >= 0.5
ORDER BY score DESC, q.q;
```

ΑΠΟΤΕΛΕΣΜΑ

```
         q         |          name          | score
-------------------+------------------------+-------
 ακουστικα         | Ακουστικά ANC          |  1.00
 ακουστικα anc     | Ακουστικά ANC          |  1.00
 δωροκαρτα         | Δωροκάρτα 50€          |  1.00
 καφετιερα         | Καφετιέρα espresso     |  1.00
 πληκτρολογιο      | Μηχανικό πληκτρολόγιο  |  1.00
 laptop            | Laptop Aero 14         |  1.00
 python            | Μαθαίνω Python         |  1.00
 postgres          | PostgreSQL σε βάθος    |  0.89
 οθονη 4k          | Οθόνη 27" 4K           |  0.75
 postgresql βιβλιο | PostgreSQL σε βάθος    |  0.61
 καρεκλα γραφειου  | Φωτιστικό γραφείου LED |  0.53
(11 rows)
```

Το `LATERAL` του κεφαλαίου 14 κρατά το καλύτερο προϊόν για κάθε αναζήτηση. Η αναζήτηση `λαπτοπ` δεν ταιριάζει με κανένα `Laptop`, αφού τα trigrams συγκρίνουν χαρακτήρες και το ελληνικό π δεν μοιάζει με το λατινικό p.

Η αναζήτηση `καρεκλα γραφειου` κατέληξε στο φωτιστικό. Το `word_similarity` ψάχνει το κομμάτι του ονόματος που μοιάζει περισσότερο με όλη την αναζήτηση και το `γραφειου` είναι μεγαλύτερο κοινό κομμάτι από το `καρεκλα`. Αν η αναζήτηση έχει πολλές λέξεις, το full text search της επόμενης ενότητας ή η αναζήτηση λέξη προς λέξη δίνουν καλύτερα αποτελέσματα. Τα trigrams είναι εργαλείο για λάθη μέσα σε μια λέξη.

Άσκηση 28.4 · Διπλοεγγραφές πελατών

Βρες ζεύγη πελατών με ομοιότητα ονοματεπωνύμου τουλάχιστον 0.6, αφού μετατρέψεις τα ελληνικά γράμματα σε λατινικά με `translate(lower(unaccent(όνομα)), 'αβγδεζηθικλμνξοπρστυφχψω', 'abgdezitiklmnxoprsstyfhpo')`. Κάθε ζεύγος μία φορά.

ΛΥΣΗ

```
WITH names AS (
  SELECT id,
         translate(lower(unaccent(first_name || ' ' || last_name)),
                   'αβγδεζηθικλμνξοπρστυφχψω', 'abgdezitiklmnxoprsstyfhpo') AS latin
  FROM customers
)
SELECT a.id, a.latin, b.id, b.latin, round(similarity(a.latin, b.latin)::numeric, 2) AS score
FROM names AS a
JOIN names AS b ON b.id > a.id
WHERE similarity(a.latin, b.latin) >= 0.6
ORDER BY score DESC;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id |        latin          | id |        latin          | score
----+-----------------------+----+-----------------------+-------
  2 | giorgos papadopoylos | 21 | giorgos papadopoulos |  0.74
(1 row)
```

Ο πελάτης 2 και ο 21 είναι το ίδιο πρόσωπο γραμμένο σε δύο αλφάβητα. Το self join του κεφαλαίου 9 τους βρήκε από την ημερομηνία γέννησης. Εδώ τους βρίσκει από το όνομα. Η μεταγραφή δεν είναι τέλεια, αλλά αρκεί για να ανεβάσει την ομοιότητα πάνω από το όριο.

Άσκηση 28.5 · Κριτικές για καφέ

Με full text search, βρες τις κριτικές που αναφέρουν καφετιέρα ή espresso, ταξινομημένες κατά `ts_rank`. Εμφάνισε `id`, τίτλο και βαθμό με τρία δεκαδικά.

ΛΥΣΗ

```
SELECT r.id, r.title,
       round(ts_rank(to_tsvector('greek', r.title || ' ' || r.body), q)::numeric, 3) AS rank
FROM reviews AS r,
     websearch_to_tsquery('greek', 'καφετιέρα or espresso') AS q
WHERE to_tsvector('greek', r.title || ' ' || r.body) @@ q
ORDER BY rank DESC, r.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id |         title          | rank
----+------------------------+-------
  6 | Espresso σαν καφετέρια | 0.068
 11 | Διαρροή                | 0.030
(2 rows)
```

Η λέξη `espresso` δεν είναι ελληνική, αλλά η ελληνική ρύθμιση την κρατά αυτούσια. Η λέξη `καφετιερα` χωρίς τόνους σε μία από τις κριτικές έγινε το ίδιο lexeme με την `καφετιέρα`.

Άσκηση 28.6 · Αποσπάσματα για ήχο

Βρες τις κριτικές που περιέχουν τη λέξη ήχος σε οποιαδήποτε μορφή και εμφάνισε `id` και απόσπασμα του σώματος με τις λέξεις ανάμεσα σε `[` και `]`.

ΛΥΣΗ

```
SELECT r.id,
       ts_headline('greek', r.body, q, 'StartSel=[, StopSel=]') AS snippet
FROM reviews AS r,
     websearch_to_tsquery('greek', 'ήχος') AS q
WHERE to_tsvector('greek', r.body) @@ q
ORDER BY r.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id |                              snippet
----+-----------------------------------------------------------------
  2 | Καθαρός [ήχος] στο podcast μας χωρίς επιπλέον κάρτα [ήχου].
 13 | Το ηχείο αντέχει νερό και άμμο. Δυνατός [ήχος] για το μέγεθός του.
 23 | Εξαιρετικος [ηχος] και μπαταρια που κραταει πολυ.
 37 | Ο [ήχος] καλός, η εφαρμογή του κινητού όμως είναι κακή.
(4 rows)
```

Το `ts_headline` σημαδεύει τις λέξεις του κειμένου που ταιριάζουν με το lexeme, όποια μορφή κι αν έχουν. Για μεγάλα κείμενα επιστρέφει μόνο τα κομμάτια γύρω από τα ταιριάσματα.

ΚΕΦΑΛΑΙΟ 29

Πώς αποθηκεύει τα δεδομένα η PostgreSQL

Άσκηση 29.1 · Μέγεθος ανά γραμμή

Για κάθε πίνακα του schema `lab` εμφάνισε όνομα, πλήθος γραμμών από τον κατάλογο και μέσο μέγεθος γραμμής σε bytes, δηλαδή μέγεθος πίνακα διά γραμμές, ως ακέραιο. Ταξινόμησε από το μεγαλύτερο μέσο μέγεθος.

ΛΥΣΗ

```
SELECT relname,
       reltuples::bigint AS rows,
       (pg_table_size(oid) / reltuples)::integer AS bytes_per_row
FROM pg_class
WHERE relnamespace = 'lab'::regnamespace AND relkind = 'r' AND reltuples > 0
ORDER BY bytes_per_row DESC;
```

ΑΠΟΤΕΛΕΣΜΑ

```
   relname   |  rows   | bytes_per_row
-------------+---------+---------------
 products    |   50000 |           184
 customers   |  200000 |           133
 events      | 1000000 |           112
 orders_copy |  100000 |           107
 orders      | 2000000 |            69
(5 rows)
```

Τα events έχουν τη μεγαλύτερη γραμμή, λόγω του jsonb. Το μέγεθος ανά γραμμή περιλαμβάνει και το κόστος της δομής, περίπου 24 bytes header για κάθε έκδοση γραμμής. Γι' αυτό ένας πίνακας με λίγες μικρές στήλες έχει πάντα μεγαλύτερο μέγεθος από το άθροισμα των στηλών του.

Άσκηση 29.2 · Η διεύθυνση μιας γραμμής

Βρες σε ποια σελίδα του `lab.orders` βρίσκονται οι παραγγελίες 500.000 και 2.000.000. Το `ctid` μετατρέπεται σε κείμενο και το πρώτο του μέρος είναι η σελίδα.

ΛΥΣΗ

```
SELECT id, ctid,
       split_part(trim(both '()' FROM ctid::text), ',', 1)::integer AS page
FROM lab.orders
WHERE id IN (500000, 2000000);
```

ΑΠΟΤΕΛΕΣΜΑ

```
   id    |    ctid    | page
---------+------------+-------
  500000 | (4231,8)   |  4231
 2000000 | (16923,28) | 16923
(2 rows)
```

Η σελίδα της τελευταίας παραγγελίας είναι η τελευταία του πίνακα, κοντά στο `relpages` του καταλόγου. Η γραμμή θα άλλαζε σελίδα με το πρώτο `UPDATE` που δεν χωρά στην ίδια σελίδα.

Άσκηση 29.3 · Διαγραφές και χώρος

Στο αντίγραφο `lab.orders_copy` της θεωρίας, διέγραψε τις ακυρωμένες παραγγελίες και μέτρησε με `pgstattuple` ζωντανές και νεκρές γραμμές. Μετά τρέξε `VACUUM` και μέτρησε ξανά.

ΛΥΣΗ

```
DELETE FROM lab.orders_copy WHERE status = 'cancelled';
SELECT tuple_count AS live, dead_tuple_count AS dead FROM pgstattuple('lab.orders_copy');
VACUUM lab.orders_copy;
SELECT tuple_count AS live, dead_tuple_count AS dead FROM pgstattuple('lab.orders_copy');
```

ΑΠΟΤΕΛΕΣΜΑ

```
DELETE 7070

 live  | dead
-------+------
 92930 | 7070
(1 row)

VACUUM

 live  | dead
-------+------
 92930 |    0
(1 row)
```

Το `DELETE` δεν μείωσε τον χώρο, απλώς σημάδεψε γραμμές ως νεκρές. Το `VACUUM` τις καθάρισε. Η λύση τρέχει εκτός transaction, αφού το `VACUUM` δεν επιτρέπεται μέσα σε transaction block.

Άσκηση 29.4 · Πόσο συμπιέζεται το JSON

Για τα events με `id` από 1 έως 5, σύγκρινε το μέγεθος του `payload` ως κείμενο με το μέγεθος που πιάνει ως `jsonb` στον δίσκο, με `octet_length` και `pg_column_size`.

ΛΥΣΗ

```
SELECT id, payload,
       octet_length(payload::text) AS text_bytes,
       pg_column_size(payload) AS jsonb_bytes
FROM lab.events
WHERE id <= 5;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id |                   payload                    | text_bytes | jsonb_bytes
----+----------------------------------------------+------------+-------------
  1 | {"device": "mobile", "product_id": 29433}    |         41 |          55
  2 | {"device": "mobile", "product_id": 36554}    |         41 |          55
  3 | {"device": "tablet", "product_id": 16764}    |         41 |          55
  4 | {"device": "desktop", "product_id": 49381}   |         42 |          55
  5 | {"device": "tablet", "product_id": 3389}     |         40 |          53
(5 rows)
```

Για μικρά έγγραφα το jsonb πιάνει περισσότερο χώρο από το κείμενο, επειδή κρατά και τη δομή για γρήγορη πρόσβαση στα κλειδιά. Το TOAST και η συμπίεση μπαίνουν σε λειτουργία μόνο όταν μια γραμμή πλησιάσει τα 2 KB.

Άσκηση 29.5 · Fillfactor και μέγεθος

Φτιάξε δύο αντίγραφα των πρώτων 20.000 παραγγελιών του `lab.orders`, το `lab.ff100` με την προεπιλογή και το `lab.ff70` με `fillfactor` 70. Μέτρησε πόσες διαφορετικές σελίδες χρησιμοποιούν οι γραμμές καθενός, από το `ctid` όπως στην άσκηση 29.2.

ΛΥΣΗ

```
CREATE TABLE lab.ff100 AS SELECT * FROM lab.orders WHERE id <= 20000;
CREATE TABLE lab.ff70 (LIKE lab.orders) WITH (fillfactor = 70);
INSERT INTO lab.ff70 SELECT * FROM lab.orders WHERE id <= 20000;

SELECT 'ff100' AS table_name,
       count(DISTINCT split_part(trim(both '()' FROM ctid::text), ',', 1)) AS used_pages
FROM lab.ff100
UNION ALL
SELECT 'ff70',
       count(DISTINCT split_part(trim(both '()' FROM ctid::text), ',', 1))
FROM lab.ff70
ORDER BY table_name;
```

ΑΠΟΤΕΛΕΣΜΑ

```
SELECT 20000

CREATE TABLE

INSERT 0 20000

 table_name | used_pages
------------+------------
 ff100      |        170
 ff70       |        243
(2 rows)
```

Ο πίνακας με fillfactor 70 χρησιμοποιεί περίπου 40% περισσότερες σελίδες, επειδή κάθε σελίδα γεμίζει μόνο ως το 70%. Η λύση μετρά σελίδες από το `ctid` και όχι από το μέγεθος του αρχείου. Όταν φορτώνεται πολλή πληροφορία μαζί, η PostgreSQL μεγαλώνει το αρχείο κατά πολλές σελίδες τη φορά, οπότε το `pg_relation_size` μπορεί να περιέχει σελίδες που δεν έχουν γεμίσει ακόμη. Ο χώρος που μένει είναι για HOT updates. Είναι μια συναλλαγή. Περισσότερος χώρος και αργότερα sequential scans για λιγότερη δουλειά στα indexes σε κάθε update.

ΚΕΦΑΛΑΙΟ 30

B-tree indexes

Άσκηση 30.1 · Αναζήτηση προϊόντος με όνομα

Φτιάξε index στο `name` του `lab.products` και δείξε με `EXPLAIN (COSTS OFF)` ότι χρησιμοποιείται για αναζήτηση με ισότητα στο προϊόν `Οθόνη Delta ECC-103`.

ΛΥΣΗ

```
CREATE INDEX products_name_idx ON lab.products (name);

EXPLAIN (COSTS OFF)
SELECT id, price FROM lab.products WHERE name = 'Οθόνη Delta ECC-103';
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE INDEX

Index Scan using products_name_idx on products
  Index Cond: (name = 'Οθόνη Delta ECC-103'::text)
```

Ένα index σε στήλη κειμένου με ελληνικό collation εξυπηρετεί ισότητα, διαστήματα και ταξινόμηση με το ίδιο collation.

Άσκηση 30.2 · Προθέματα και collation

Με το index της προηγούμενης άσκησης, δείξε με `EXPLAIN` ότι το `name LIKE 'Καφετιέρα Aero%'` δεν το χρησιμοποιεί. Φτιάξε δεύτερο index με `text_pattern_ops` και δείξε ότι τώρα χρησιμοποιείται.

ΛΥΣΗ

```
CREATE INDEX products_name_idx ON lab.products (name);

EXPLAIN (COSTS OFF)
SELECT id FROM lab.products WHERE name LIKE 'Καφετιέρα Aero%';

CREATE INDEX products_name_pattern_idx ON lab.products (name text_pattern_ops);

EXPLAIN (COSTS OFF)
SELECT id FROM lab.products WHERE name LIKE 'Καφετιέρα Aero%';
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE INDEX

Seq Scan on products
  Filter: (name ~~ 'Καφετιέρα Aero%'::text)

CREATE INDEX

Bitmap Heap Scan on products
  Filter: (name ~~ 'Καφετιέρα Aero%'::text)
  ->  Bitmap Index Scan on products_name_pattern_idx
        Index Cond: ((name ~>=~ 'Καφετιέρα Aero'::text) AND (name ~<~ 'Καφετιέρα Aerp'::text))
```

Με γλωσσικό collation η σειρά ταξινόμησης δεν είναι σειρά χαρακτήρων, οπότε ένα πρόθεμα δεν αντιστοιχεί σε συνεχόμενο κομμάτι του index. Το `text_pattern_ops` φτιάχνει index σε σειρά χαρακτήρων, κατάλληλο για `LIKE 'πρόθεμα%'`. Για `LIKE '%κομμάτι%'` κανένα B-tree δεν βοηθά. Εκεί χρειάζονται τα trigram indexes του κεφαλαίου 35.

Άσκηση 30.3 · Τα τελευταία events ενός πελάτη

Φτιάξε το index που επιτρέπει να βρίσκονται τα τρία πιο πρόσφατα events του πελάτη 777 στον `lab.events` χωρίς ταξινόμηση. Δείξε το plan.

ΛΥΣΗ

```
CREATE INDEX events_customer_time_idx ON lab.events (customer_id, occurred_at);

EXPLAIN (COSTS OFF)
SELECT id, kind, occurred_at
FROM lab.events
WHERE customer_id = 777
ORDER BY occurred_at DESC
LIMIT 3;
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE INDEX

Limit
  ->  Index Scan Backward using events_customer_time_idx on events
        Index Cond: (customer_id = 777)
```

Το plan δεν έχει κόμβο `Sort`. Το index δίνει τις γραμμές ήδη ταξινομημένες και το `Limit` σταματά μετά από τρεις. Το ίδιο index με φθίνουσα σειρά, `(customer_id, occurred_at DESC)`, θα δούλευε το ίδιο, αφού ένα B-tree διαβάζεται και προς τις δύο κατευθύνσεις.

Άσκηση 30.4 · Sargable μήνας

Το query `SELECT count(*) FROM lab.events WHERE extract(year FROM occurred_at) = 2025 AND extract(month FROM occurred_at) = 3` διαβάζει όλο τον πίνακα. Φτιάξε index στο `occurred_at` και ξαναγράψε το query ώστε να τον χρησιμοποιεί. Δείξε το plan και το αποτέλεσμα.

ΛΥΣΗ

```
CREATE INDEX events_time_idx ON lab.events (occurred_at);

EXPLAIN (COSTS OFF)
SELECT count(*) FROM lab.events
WHERE occurred_at >= '2025-03-01' AND occurred_at < '2025-04-01';

SELECT count(*) FROM lab.events
WHERE occurred_at >= '2025-03-01' AND occurred_at < '2025-04-01';
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE INDEX

Aggregate
  ->  Index Only Scan using events_time_idx on events
        Index Cond: ((occurred_at >= '2025-03-01 00:00:00+02'::timestamp with time zone)
AND (occurred_at < '2025-04-01 00:00:00+03'::timestamp with time zone))

 count
-------
 56881
(1 row)
```

Το μισάνοιχτο διάστημα του κεφαλαίου 6 είναι και η sargable μορφή. Η στήλη μένει μόνη της και η βάση βρίσκει την αρχή του διαστήματος στο index.

Άσκηση 30.5 · Partial και unique

Στον `lab.customers`, φτιάξε unique index που εγγυάται ότι δεν υπάρχουν δύο πελάτες με το ίδιο email χωρίς διάκριση κεφαλαίων. Μετά δοκίμασε να προσθέσεις πελάτη 200001 με email `USER1@EXAMPLE.GR`.

ΛΥΣΗ

```
CREATE UNIQUE INDEX customers_email_ci_idx ON lab.customers (lower(email));

INSERT INTO lab.customers (id, first_name, last_name, email, city, region, created_at)
VALUES (200001, 'Νέος', 'Πελάτης', 'USER1@EXAMPLE.GR', 'Αθήνα', 'Αττική', '2026-06-30');
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE INDEX

ERROR:  duplicate key value violates unique constraint "customers_email_ci_idx"
DETAIL:  Key (lower(email))=(user1@example.gr) already exists.
```

Το email υπάρχει ήδη σε πεζά για τον πελάτη 1. Το unique index στην έκφραση το εντοπίζει. Ένα `UNIQUE (email)` constraint δεν θα το έβλεπε.

Άσκηση 30.6 · Index μόνο για ανάγνωση

Φτιάξε index που επιτρέπει `Index Only Scan` για το query που μετρά τα events ανά `kind` για τον Μάρτιο του 2025. Κάνε `VACUUM` στον πίνακα και δείξε το plan.

ΛΥΣΗ

```
CREATE INDEX events_time_kind_idx ON lab.events (occurred_at) INCLUDE (kind);
VACUUM lab.events;

EXPLAIN (COSTS OFF)
SELECT kind, count(*)
FROM lab.events
WHERE occurred_at >= '2025-03-01' AND occurred_at < '2025-04-01'
GROUP BY kind;
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE INDEX

VACUUM

HashAggregate
  Group Key: kind
  ->  Index Only Scan using events_time_kind_idx on events
        Index Cond: ((occurred_at >= '2025-03-01 00:00:00+02'::timestamp with time zone)
AND (occurred_at < '2025-04-01 00:00:00+03'::timestamp with time zone))
```

Το `occurred_at` είναι η στήλη αναζήτησης και το `kind` περιλαμβάνεται για να μη χρειάζεται ο πίνακας. Η λύση τρέχει εκτός transaction επειδή το `VACUUM` δεν επιτρέπεται μέσα σε transaction block.

Στον `lab.orders` το ίδιο κόλπο δεν θα είχε αποτέλεσμα, γιατί η θεωρία έφτιαξε ήδη index στο `created_at`. Οι παραγγελίες είναι γραμμένες στον δίσκο με τη σειρά της ημερομηνίας, οπότε ένα απλό `Index Scan` διαβάζει συνεχόμενες σελίδες του πίνακα και ο planner το βρίσκει ελάχιστα φθηνότερο από ένα μεγαλύτερο covering index. Το κεφάλαιο 34 εξηγεί πώς η βάση ξέρει τη σειρά των γραμμών στον δίσκο.

ΚΕΦΑΛΑΙΟ 31

Διαβάζοντας το EXPLAIN

Άσκηση 31.1 · Υπολόγισε το κόστος

Υπολόγισε από τον κατάλογο το κόστος ενός `Seq Scan` στον `lab.products` και σύγκρινέ το με το `EXPLAIN SELECT * FROM lab.products`.

ΛΥΣΗ

```
SELECT relpages * current_setting('seq_page_cost')::numeric
     + reltuples * current_setting('cpu_tuple_cost')::numeric AS computed
FROM pg_class
WHERE oid = 'lab.products'::regclass;

EXPLAIN SELECT * FROM lab.products;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 computed
----------
     1616
(1 row)

Seq Scan on products  (cost=0.00..1616.00 rows=50000 width=147)
```

Τα δύο νούμερα είναι ίδια. Ο planner υπολογίζει το κόστος από το μέγεθος του πίνακα και τις σταθερές κόστους, χωρίς να διαβάσει δεδομένα.

Άσκηση 31.2 · Κόστος του φίλτρου

Δείξε με `EXPLAIN` το plan του `SELECT * FROM lab.products WHERE price > 800`. Υπολόγισε τη διαφορά του συνολικού κόστους από το plan της προηγούμενης άσκησης και εξήγησε από πού έρχεται.

ΛΥΣΗ

```
EXPLAIN SELECT * FROM lab.products WHERE price > 800;
```

ΑΠΟΤΕΛΕΣΜΑ

```
Seq Scan on products  (cost=0.00..1741.00 rows=2997 width=147)
  Filter: (price > '800'::numeric)
```

Το κόστος αυξήθηκε κατά 125, δηλαδή 50.000 γραμμές επί 0.0025, το κόστος της σύγκρισης ανά γραμμή που ορίζει η ρύθμιση `cpu_operator_cost`. Οι σελίδες που διαβάζονται είναι ίδιες, αφού ένα `Seq Scan` διαβάζει όλο τον πίνακα με ή χωρίς φίλτρο.

Άσκηση 31.3 · Αν δεν υπήρχε το Seq Scan

Σε transaction, κάνε `SET LOCAL enable_seqscan = off` και δείξε το plan του `SELECT sum(total) FROM lab.orders`. Σύγκρινε με το κανονικό plan και εξήγησε τι σημαίνει η νέα γραμμή που εμφανίζεται.

ΛΥΣΗ

```
EXPLAIN SELECT sum(total) FROM lab.orders;
SET LOCAL enable_seqscan = off;
EXPLAIN SELECT sum(total) FROM lab.orders;
```

ΑΠΟΤΕΛΕΣΜΑ

```
Aggregate  (cost=41924.00..41924.01 rows=1 width=32)
  ->  Seq Scan on orders  (cost=0.00..36924.00 rows=2000000 width=6)

SET

Aggregate  (cost=41924.00..41924.01 rows=1 width=32)
  ->  Seq Scan on orders  (cost=0.00..36924.00 rows=2000000 width=6)
        Disabled: true
```

Το plan έμεινε ίδιο, με τη σημείωση `Disabled: true`. Ο planner δεν είχε εναλλακτική. Κανένα index δεν περιέχει το `total` ούτε βοηθά κάποια συνθήκη ή ταξινόμηση. Το `enable_seqscan = off` δεν απαγορεύει το `Seq Scan`. Λέει στον planner να το αποφύγει όταν μπορεί. Από την PostgreSQL 18 το plan δείχνει ποιοι κόμβοι επιλέχθηκαν παρόλο που ήταν απενεργοποιημένοι. Το `SET LOCAL` ισχύει μόνο μέχρι το τέλος του transaction. Η λύση εκτελέστηκε μέσα σε transaction που αναιρέθηκε.

Άσκηση 31.4 · Το plan ενός anti join

Δείξε με `EXPLAIN (COSTS OFF)` το plan του query που βρίσκει τα προϊόντα του `lab.products` χωρίς κανένα event, γραμμένο με `NOT EXISTS` πάνω στο `(payload ->> 'product_id')::integer`.

ΛΥΣΗ

```
EXPLAIN (COSTS OFF)
SELECT p.id
FROM lab.products AS p
WHERE NOT EXISTS (SELECT 1 FROM lab.events AS e
                  WHERE (e.payload ->> 'product_id')::integer = p.id);
```

ΑΠΟΤΕΛΕΣΜΑ

```
Hash Right Anti Join
  Hash Cond: (((e.payload ->> 'product_id'::text))::integer = p.id)
  ->  Seq Scan on events e
  ->  Hash
        ->  Index Only Scan using products_pkey on products p
```

Το plan δεν εκτελεί το subquery για κάθε προϊόν. Ο κόμβος `Hash Right Anti Join` ενώνει τους δύο πίνακες μία φορά και κρατά τα προϊόντα χωρίς ταίρι. Είναι η μετατροπή σε anti join που ανέφερε το κεφάλαιο 13.

Άσκηση 31.5 · Ποιες στήλες κουβαλά κάθε κόμβος

Με `EXPLAIN (VERBOSE, COSTS OFF)`, δείξε το plan του `SELECT city, count(*) FROM lab.customers GROUP BY city`. Εντόπισε ποιες στήλες διαβάζει το `Seq Scan`.

ΛΥΣΗ

```
EXPLAIN (VERBOSE, COSTS OFF)
SELECT city, count(*) FROM lab.customers GROUP BY city;
```

ΑΠΟΤΕΛΕΣΜΑ

```
HashAggregate
  Output: city, count(*)
  Group Key: customers.city
  ->  Seq Scan on lab.customers
        Output: id, first_name, last_name, email, city, region, created_at
Query Identifier: 4132498954773243860
```

Το `Output` του `Seq Scan` έχει όλες τις στήλες του πίνακα, αν και χρειάζεται μόνο η `city`. Είναι βελτιστοποίηση. Όταν ο γονέας δεν απαιτεί κάτι άλλο, ο κόμβος περνά τη γραμμή όπως είναι στη σελίδα, αντί να φτιάξει νέα γραμμή με μία στήλη. Το `HashAggregate` από πάνω βγάζει μόνο τις δύο στήλες του αποτελέσματος. Η γραμμή `Query Identifier` εμφανίζεται όταν είναι ενεργή η επέκταση `pg_stat_statements`, που θα δούμε στο κεφάλαιο 36.

ΚΕΦΑΛΑΙΟ 32

EXPLAIN ANALYZE και BUFFERS

Άσκηση 32.1 · Πόσες σελίδες για λίγες γραμμές

Με `EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)`, δείξε το plan για τα events του πελάτη 4242. Φτιάξε index στο `customer_id` του `lab.events` και δείξε το plan ξανά. Σύγκρινε τις σελίδες.

ΛΥΣΗ

```
EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)
SELECT id, kind FROM lab.events WHERE customer_id = 4242;

CREATE INDEX events_customer_idx ON lab.events (customer_id);

EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)
SELECT id, kind FROM lab.events WHERE customer_id = 4242;
```

ΑΠΟΤΕΛΕΣΜΑ

```
Seq Scan on events (actual rows=11.00 loops=1)
  Filter: (customer_id = 4242)
  Rows Removed by Filter: 999989
  Buffers: shared read=13659
Planning:
  Buffers: shared hit=5 read=1 dirtied=1
Planning Time: 0.127 ms
Execution Time: 34.468 ms

CREATE INDEX

Bitmap Heap Scan on events (actual rows=11.00 loops=1)
  Recheck Cond: (customer_id = 4242)
  Heap Blocks: exact=11
  Buffers: shared read=14
  ->  Bitmap Index Scan on events_customer_idx (actual rows=11.00 loops=1)
        Index Cond: (customer_id = 4242)
        Index Searches: 1
        Buffers: shared read=3
Planning:
  Buffers: shared hit=4 read=1
Planning Time: 0.120 ms
Execution Time: 0.138 ms
```

Χωρίς index η βάση διάβασε όλες τις σελίδες του πίνακα για λίγες γραμμές. Με index διάβασε μερικές σελίδες του index και μία σελίδα του πίνακα για κάθε γραμμή, αφού τα events ενός πελάτη είναι σκορπισμένα.

Άσκηση 32.2 · Υπολόγισε από τα loops

Με `EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)`, δείξε το plan για τις παραγγελίες των πελατών 150000 έως 150100, με join ανάμεσα στον `lab.customers` και στον `lab.orders`. Από τα `rows` και τα `loops` του εσωτερικού κόμβου υπολόγισε πόσες παραγγελίες βρέθηκαν συνολικά και έλεγξέ το με `count(*)`.

ΛΥΣΗ

```
EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)
SELECT o.id
FROM lab.customers AS c
JOIN lab.orders AS o ON o.customer_id = c.id
WHERE c.id BETWEEN 150000 AND 150100;

SELECT count(*)
FROM lab.customers AS c
JOIN lab.orders AS o ON o.customer_id = c.id
WHERE c.id BETWEEN 150000 AND 150100;
```

ΑΠΟΤΕΛΕΣΜΑ

```
Nested Loop (actual rows=586.00 loops=1)
  Buffers: shared hit=320 read=574
  ->  Index Only Scan using customers_pkey on customers c (actual rows=101.00 loops=1)
        Index Cond: ((id >= 150000) AND (id <= 150100))
        Heap Fetches: 0
        Index Searches: 1
        Buffers: shared hit=3 read=2
  ->  Index Scan using orders_customer_idx on orders o (actual rows=5.80 loops=101)
        Index Cond: (customer_id = c.id)
        Index Searches: 101
        Buffers: shared hit=317 read=572
Planning:
  Buffers: shared hit=20 read=10
Planning Time: 0.245 ms
Execution Time: 1.934 ms

 count
-------
   586
(1 row)
```

Το εσωτερικό σκέλος εκτελέστηκε μία φορά για κάθε έναν από τους 101 πελάτες. Τα `rows` του είναι ο μέσος όρος παραγγελιών ανά πελάτη. Το γινόμενό τους με τα `loops` δίνει το ίδιο με το `count(*)`.

Άσκηση 32.3 · Αρκετή μνήμη για ένα hash

Δείξε με `EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)` το plan του `SELECT customer_id, count(*) FROM lab.events GROUP BY customer_id`. Μετά, μέσα στο ίδιο transaction, κάνε `SET LOCAL work_mem = '64MB'` και δείξε το ξανά. Βρες τι άλλαξε.

ΛΥΣΗ

```
EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)
SELECT customer_id, count(*) FROM lab.events GROUP BY customer_id;

SET LOCAL work_mem = '64MB';

EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)
SELECT customer_id, count(*) FROM lab.events GROUP BY customer_id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
HashAggregate (actual rows=198658.00 loops=1)
  Group Key: customer_id
  Batches: 5  Memory Usage: 8249kB  Disk Usage: 11424kB
  Buffers: shared hit=11 read=13648, temp read=1210 written=2325
  ->  Seq Scan on events (actual rows=1000000.00 loops=1)
        Buffers: shared hit=11 read=13648
Planning:
  Buffers: shared hit=8 dirtied=2
Planning Time: 0.055 ms
Execution Time: 237.733 ms

SET

HashAggregate (actual rows=198658.00 loops=1)
  Group Key: customer_id
  Batches: 1  Memory Usage: 12313kB
  Buffers: shared read=13659
  ->  Seq Scan on events (actual rows=1000000.00 loops=1)
        Buffers: shared read=13659
Planning Time: 0.075 ms
Execution Time: 146.586 ms
```

Με λίγη μνήμη το `HashAggregate` χώρισε τις ομάδες σε κομμάτια και έγραψε στον δίσκο. Το δείχνουν τα `Batches` και το `Disk Usage`. Με περισσότερη μνήμη όλες οι ομάδες χωρούν μαζί και το `Batches` γίνεται 1.

Άσκηση 32.4 · Μια εκτίμηση που αστοχεί

Με `EXPLAIN ANALYZE`, σύγκρινε την εκτίμηση και την πραγματικότητα για τους πελάτες με `city = 'Αθήνα' AND region = 'Κρήτη'`, ένα συνδυασμό που δεν υπάρχει.

ΛΥΣΗ

```
EXPLAIN ANALYZE
SELECT * FROM lab.customers WHERE city = 'Αθήνα' AND region = 'Κρήτη';
```

ΑΠΟΤΕΛΕΣΜΑ

```
Seq Scan on customers  (cost=0.00..6249.00 rows=9544 width=98) (actual time=13.977..13.977 rows=0.00 loops=1)
  Filter: ((city = 'Αθήνα'::text) AND (region = 'Κρήτη'::text))
  Rows Removed by Filter: 200000
  Buffers: shared read=3249 written=174
Planning Time: 0.100 ms
Execution Time: 13.984 ms
```

Ο planner περιμένει χιλιάδες γραμμές και βρίσκει μηδέν. Το γινόμενο των δύο πιθανοτήτων είναι αρκετά μεγάλο, αλλά ο συνδυασμός είναι αδύνατος. Είναι το ίδιο πρόβλημα με τη θεωρία, στην αντίθετη κατεύθυνση.

Άσκηση 32.5 · EXPLAIN ANALYZE χωρίς συνέπειες

Μέσα σε transaction, δείξε το `EXPLAIN ANALYZE` ενός `UPDATE` που κάνει `delivered` τις παραγγελίες `shipped` του `lab.orders`. Μετά το `ROLLBACK` έλεγξε ότι το πλήθος των `shipped` δεν άλλαξε.

ΛΥΣΗ

```
BEGIN;
EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)
UPDATE lab.orders SET status = 'delivered' WHERE status = 'shipped';
ROLLBACK;
SELECT count(*) AS shipped FROM lab.orders WHERE status = 'shipped';
```

ΑΠΟΤΕΛΕΣΜΑ

```
BEGIN

Update on orders (actual rows=0.00 loops=1)
  Buffers: shared hit=17573 read=17997 dirtied=1135 written=37
  ->  Seq Scan on orders (actual rows=1330.00 loops=1)
        Filter: (status = 'shipped'::text)
        Rows Removed by Filter: 1998670
        Buffers: shared read=16924
Planning:
  Buffers: shared read=3 dirtied=1
Planning Time: 0.222 ms
Execution Time: 72.770 ms

ROLLBACK

 shipped
---------
    1330
(1 row)
```

Το plan δείχνει έναν κόμβο `Update` πάνω από το `Seq Scan`. Το `UPDATE` έγινε πραγματικά και μετά αναιρέθηκε. Χωρίς το `BEGIN`, το `EXPLAIN ANALYZE` θα είχε αλλάξει τα δεδομένα.

ΚΕΦΑΛΑΙΟ 33

Joins, aggregates και parallel query στον planner

Άσκηση 33.1 · Από nested loop σε hash join

Με `EXPLAIN (COSTS OFF)`, δείξε το plan για τις παραγγελίες των πελατών με `id` από 150000 έως 150010 και μετά από 1 έως 50000. Εξήγησε γιατί άλλαξε ο αλγόριθμος.

ΛΥΣΗ

```
EXPLAIN (COSTS OFF)
SELECT o.id FROM lab.customers AS c
JOIN lab.orders AS o ON o.customer_id = c.id
WHERE c.id BETWEEN 150000 AND 150010;

EXPLAIN (COSTS OFF)
SELECT o.id FROM lab.customers AS c
JOIN lab.orders AS o ON o.customer_id = c.id
WHERE c.id BETWEEN 1 AND 50000;
```

ΑΠΟΤΕΛΕΣΜΑ

```
Nested Loop
  ->  Index Only Scan using customers_pkey on customers c
        Index Cond: ((id >= 150000) AND (id <= 150010))
  ->  Bitmap Heap Scan on orders o
        Recheck Cond: (c.id = customer_id)
        ->  Bitmap Index Scan on orders_customer_idx
              Index Cond: (customer_id = c.id)

Hash Join
  Hash Cond: (o.customer_id = c.id)
  ->  Seq Scan on orders o
  ->  Hash
        ->  Index Only Scan using customers_pkey on customers c
              Index Cond: ((id >= 1) AND (id <= 50000))
```

Για 11 πελάτες, 11 αναζητήσεις στο index είναι φθηνότερες από την ανάγνωση όλων των παραγγελιών. Για 50.000 πελάτες, που έχουν μάλιστα τις περισσότερες παραγγελίες, οι αναζητήσεις θα ήταν πάρα πολλές. Ο planner διαβάζει όλες τις παραγγελίες μία φορά και κάνει hash join, με παραλληλισμό.

Άσκηση 33.2 · Batches σε hash join

Μέσα σε transaction με `SET LOCAL work_mem = '1MB'` και χωρίς παραλληλισμό, δείξε με `EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)` το join όλων των παραγγελιών με όλους τους πελάτες που μετρά παραγγελίες ανά περιφέρεια. Εντόπισε το πλήθος των batches.

ΛΥΣΗ

```
SET LOCAL work_mem = '1MB';
SET LOCAL max_parallel_workers_per_gather = 0;

EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)
SELECT c.region, count(*)
FROM lab.orders AS o
JOIN lab.customers AS c ON c.id = o.customer_id
GROUP BY c.region;
```

ΑΠΟΤΕΛΕΣΜΑ

```
SET

SET

HashAggregate (actual rows=9.00 loops=1)
  Group Key: c.region
  Batches: 1  Memory Usage: 32kB
  Buffers: shared read=20173 written=1963, temp read=6021 written=6021
  ->  Hash Join (actual rows=2000000.00 loops=1)
        Hash Cond: (o.customer_id = c.id)
        Buffers: shared read=20173 written=1963, temp read=6021 written=6021
        ->  Seq Scan on orders o (actual rows=2000000.00 loops=1)
              Buffers: shared read=16924 written=36
        ->  Hash (actual rows=200000.00 loops=1)
              Buckets: 32768  Batches: 8  Memory Usage: 1674kB
              Buffers: shared read=3249 written=1927, temp written=962
              ->  Seq Scan on customers c (actual rows=200000.00 loops=1)
                    Buffers: shared read=3249 written=1927
Planning:
  Buffers: shared hit=18 read=1 dirtied=1 written=1
Planning Time: 0.083 ms
Execution Time: 570.532 ms
```

Ο πίνακας κατακερματισμού των 200.000 πελάτων δεν χωρά σε 1 MB, οπότε το `Hash` χωρίστηκε σε πολλά batches. Και οι δύο είσοδοι γράφτηκαν σε προσωρινά αρχεία, όπως φαίνεται στα `temp` των buffers.

Άσκηση 33.3 · Semi join

Με `EXPLAIN (COSTS OFF)` χωρίς παραλληλισμό, δείξε το plan για το πλήθος των πελατών του `lab.customers` που έχουν τουλάχιστον μία παραγγελία πάνω από 1400 ευρώ, με `EXISTS`.

ΛΥΣΗ

```
SET LOCAL max_parallel_workers_per_gather = 0;

EXPLAIN (COSTS OFF)
SELECT count(*)
FROM lab.customers AS c
WHERE EXISTS (SELECT 1 FROM lab.orders AS o
              WHERE o.customer_id = c.id AND o.total > 1400);
```

ΑΠΟΤΕΛΕΣΜΑ

```
SET

Aggregate
  ->  Hash Join
        Hash Cond: (c.id = o.customer_id)
        ->  Index Only Scan using customers_pkey on customers c
        ->  Hash
              ->  HashAggregate
                    Group Key: o.customer_id
                    ->  Seq Scan on orders o
                          Filter: (total > '1400'::numeric)
```

Ένα semi join θα ήταν ο προφανής δρόμος, αλλά ο planner βρήκε φθηνότερο άλλον. Βρίσκει πρώτα τις παραγγελίες πάνω από 1400 ευρώ, κρατά τους μοναδικούς πελάτες τους με `HashAggregate` και κάνει κανονικό join με τον `lab.customers`. Οι ακριβές παραγγελίες είναι λίγες, οπότε η λίστα των πελατών τους είναι μικρή. Ο planner μετατρέπει ελεύθερα ένα semi join σε κανονικό join όταν μπορεί να εγγυηθεί ότι δεν θα διπλασιαστούν γραμμές.

Άσκηση 33.4 · Merge join από δύο indexes

Μέσα σε transaction, απενεργοποίησε το hash join και τον παραλληλισμό και δείξε με `EXPLAIN (COSTS OFF)` το plan για το άθροισμα των παραγγελιών ανά περιφέρεια. Εντόπισε από πού έρχεται η ταξινόμηση κάθε εισόδου.

ΛΥΣΗ

```
SET LOCAL enable_hashjoin = off;
SET LOCAL max_parallel_workers_per_gather = 0;

EXPLAIN (COSTS OFF)
SELECT c.region, sum(o.total)
FROM lab.customers AS c
JOIN lab.orders AS o ON o.customer_id = c.id
GROUP BY c.region;
```

ΑΠΟΤΕΛΕΣΜΑ

```
SET

SET

HashAggregate
  Group Key: c.region
  ->  Merge Join
        Merge Cond: (c.id = o.customer_id)
        ->  Index Scan using customers_pkey on customers c
        ->  Index Scan using orders_customer_idx on orders o
```

Οι πελάτες έρχονται ταξινομημένοι από το `customers_pkey` και οι παραγγελίες από το `orders_customer_idx`. Κανένας κόμβος `Sort` δεν χρειάζεται πριν από το merge join.

Άσκηση 33.5 · Πόσοι workers

Με `EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF, BUFFERS OFF)`, δείξε το plan για το άθροισμα όλων των `total` του `lab.orders`, πρώτα με την προεπιλογή και μετά σε transaction με `SET LOCAL max_parallel_workers_per_gather = 4`.

ΛΥΣΗ

```
EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF, BUFFERS OFF)
SELECT sum(total) FROM lab.orders;

SET LOCAL max_parallel_workers_per_gather = 4;

EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF, BUFFERS OFF)
SELECT sum(total) FROM lab.orders;
```

ΑΠΟΤΕΛΕΣΜΑ

```
Finalize Aggregate (actual rows=1.00 loops=1)
  ->  Gather (actual rows=3.00 loops=1)
        Workers Planned: 2
        Workers Launched: 2
        ->  Partial Aggregate (actual rows=1.00 loops=3)
              ->  Parallel Seq Scan on orders (actual rows=666666.67 loops=3)
Planning Time: 0.030 ms
Execution Time: 39.636 ms

SET

Finalize Aggregate (actual rows=1.00 loops=1)
  ->  Gather (actual rows=4.00 loops=1)
        Workers Planned: 3
        Workers Launched: 3
        ->  Partial Aggregate (actual rows=1.00 loops=4)
              ->  Parallel Seq Scan on orders (actual rows=500000.00 loops=4)
Planning Time: 0.029 ms
Execution Time: 31.050 ms
```

Με την προεπιλογή εκτελέστηκαν δύο workers. Με όριο τέσσερα, ο planner ζήτησε όσους κρίνει κατάλληλους για το μέγεθος του πίνακα, που μπορεί να είναι λιγότεροι από τέσσερις. Τα `Workers Planned`

και `Workers Launched` μπορεί να διαφέρουν, αν ο server δεν έχει διαθέσιμους workers τη στιγμή της εκτέλεσης.

ΚΕΦΑΛΑΙΟ 34

Statistics και εκτιμήσεις

Άσκηση 34.1 · Η κατανομή των καταστάσεων

Εμφάνισε από το `pg_stats` τις συχνές τιμές του `status` του `lab.orders` με τις συχνότητές τους. Σύγκρινέ τες με τα πραγματικά ποσοστά.

ΛΥΣΗ

```
WITH estimated AS (
  SELECT t.value, round(t.freq::numeric, 4) AS sample_freq
  FROM pg_stats AS s,
       unnest(s.most_common_vals::text::text[], s.most_common_freqs) AS t(value, freq)
  WHERE s.schemaname = 'lab' AND s.tablename = 'orders' AND s.attname = 'status'
),
actual AS (
  SELECT status, round(count(*) / 2000000.0, 4) AS real_freq
  FROM lab.orders
  GROUP BY status
)
SELECT e.value, e.sample_freq, a.real_freq
FROM estimated AS e
JOIN actual AS a ON a.status = e.value
ORDER BY a.real_freq DESC, e.value;
```

ΑΠΟΤΕΛΕΣΜΑ

```
   value   | sample_freq | real_freq
-----------+-------------+-----------
 delivered |      0.8983 |    0.8983
 cancelled |      0.0699 |    0.0697
 refunded  |      0.0299 |    0.0299
 paid      |      0.0007 |    0.0007
 pending   |      0.0007 |    0.0007
 shipped   |      0.0007 |    0.0007
(6 rows)
```

Οι συχνότητες του δείγματος είναι κοντά στις πραγματικές. Οι σπάνιες τιμές, όπως το `pending`, έχουν τη μεγαλύτερη σχετική απόκλιση, αφού εμφανίστηκαν λίγες φορές μέσα στις 30.000 γραμμές του δείγματος.

Άσκηση 34.2 · Εκτίμηση για ένα διάστημα

Με `EXPLAIN ANALYZE`, σύγκρινε εκτίμηση και πραγματικότητα για τις παραγγελίες με `total > 1400`.

ΛΥΣΗ

```
EXPLAIN ANALYZE SELECT id FROM lab.orders WHERE total > 1400;
```

ΑΠΟΤΕΛΕΣΜΑ

```
Seq Scan on orders  (cost=0.00..41924.00 rows=48505 width=8) (actual time=0.090..113.086 rows=48284.00 loops=1)
  Filter: (total > '1400'::numeric)
  Rows Removed by Filter: 1951716
  Buffers: shared hit=59 read=16865 written=35
Planning:
  Buffers: shared hit=8
Planning Time: 0.039 ms
Execution Time: 114.005 ms
```

Η εκτίμηση βγαίνει από το ιστόγραμμα της στήλης `total` και είναι κοντά στην πραγματικότητα. Τα ιστογράμματα δουλεύουν καλά για μία στήλη. Οι αστοχίες εμφανίζονται κυρίως σε συνδυασμούς στηλών και σε εκφράσεις.

Άσκηση 34.3 · Statistics για μια έκφραση

Δείξε με `EXPLAIN` την εκτίμηση για τους πελάτες του `lab.customers` με `lower(email) LIKE 'user1%'`. Φτιάξε extended statistics πάνω στην έκφραση `lower(email)`, τρέξε `ANALYZE` και δείξε ξανά την εκτίμηση. Μέτρησε και τις πραγματικές γραμμές.

ΛΥΣΗ

```
EXPLAIN SELECT id FROM lab.customers WHERE lower(email) LIKE 'user1%';

CREATE STATISTICS lab.customers_email_lower ON (lower(email)) FROM lab.customers;
ANALYZE lab.customers;

EXPLAIN SELECT id FROM lab.customers WHERE lower(email) LIKE 'user1%';
SELECT count(*) FROM lab.customers WHERE lower(email) LIKE 'user1%';
```

ΑΠΟΤΕΛΕΣΜΑ

```
Seq Scan on customers  (cost=0.00..6249.00 rows=1000 width=4)
  Filter: (lower(email) ~~ 'user1%'::text)

CREATE STATISTICS

ANALYZE

Seq Scan on customers  (cost=0.00..6249.00 rows=111111 width=4)
  Filter: (lower(email) ~~ 'user1%'::text)

 count
--------
 111111
(1 row)
```

Χωρίς statistics για την έκφραση, ο planner χρησιμοποιεί μια σταθερή υπόθεση, επειδή δεν ξέρει τίποτα για τις τιμές του `lower(email)`. Με statistics εκτιμά από ιστόγραμμα της ίδιας της έκφρασης. Ένα index πάνω στην έκφραση μαζεύει επίσης τέτοια statistics.

Άσκηση 34.4 · Δύο συνθήκες που σχετίζονται

Στον `lab.orders`, οι παραγγελίες `pending` είναι όλες από τις τελευταίες ημέρες. Σύγκρινε με `EXPLAIN ANALYZE` εκτίμηση και πραγματικότητα για το `status = 'pending' AND created_at >= '2026-06-25'`. Εξήγησε την απόκλιση.

ΛΥΣΗ

```
EXPLAIN ANALYZE
SELECT id FROM lab.orders
WHERE status = 'pending' AND created_at >= '2026-06-25';
```

ΑΠΟΤΕΛΕΣΜΑ

```
Seq Scan on orders  (cost=0.00..46924.00 rows=4 width=8) (actual time=78.475..78.705 rows=1349.00 loops=1)
  Filter: ((created_at >= '2026-06-25 00:00:00+03'::timestamp with time zone) AND (status = 'pending'::text))
  Rows Removed by Filter: 1998651
  Buffers: shared hit=123 read=16801 written=22
Planning:
  Buffers: shared hit=6 read=1
Planning Time: 0.045 ms
Execution Time: 78.737 ms
```

Ο planner πολλαπλασίασε την πιθανότητα του `pending` με την πιθανότητα της ημερομηνίας, σαν να ήταν ανεξάρτητες. Στην πράξη οι pending παραγγελίες είναι σχεδόν όλες μέσα στο διάστημα, οπότε η πραγματικότητα είναι πολλαπλάσια της εκτίμησης. Τα `dependencies` του `CREATE STATISTICS` δεν βοηθούν εδώ, γιατί αφορούν ισότητες. Αν μια τέτοια υποεκτίμηση οδηγούσε σε κακό plan, η λύση θα ήταν ένα partial index όπως στο κεφάλαιο 30.

Άσκηση 34.5 · Φόρτωση και ANALYZE

Φτιάξε κενό αντίγραφο του `lab.customers`, τρέξε `ANALYZE` και φόρτωσε τους πελάτες της Αθήνας. Δείξε με `EXPLAIN` την εκτίμηση για το `WHERE city = 'Θεσσαλονίκη'` πριν και μετά από νέο `ANALYZE`.

ΛΥΣΗ

```
CREATE TABLE lab.customers_copy (LIKE lab.customers);
ANALYZE lab.customers_copy;
INSERT INTO lab.customers_copy SELECT * FROM lab.customers WHERE city = 'Αθήνα';

EXPLAIN SELECT * FROM lab.customers_copy WHERE city = 'Θεσσαλονίκη';
ANALYZE lab.customers_copy;
EXPLAIN SELECT * FROM lab.customers_copy WHERE city = 'Θεσσαλονίκη';
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE TABLE

ANALYZE

INSERT 0 80043

Seq Scan on customers_copy  (cost=0.00..1812.00 rows=242 width=172)
  Filter: (city = 'Θεσσαλονίκη'::text)

ANALYZE

Seq Scan on customers_copy  (cost=0.00..2208.54 rows=1 width=89)
  Filter: (city = 'Θεσσαλονίκη'::text)
```

Πριν από το `ANALYZE`, ο planner περίμενε μερικές εκατοντάδες γραμμές, αφού δεν ήξερε ότι όλοι οι πελάτες είναι από την Αθήνα. Μετά το `ANALYZE` ξέρει ότι η Θεσσαλονίκη δεν υπάρχει στη λίστα και εκτιμά μία γραμμή.

ΚΕΦΑΛΑΙΟ 35

GIN, GiST και BRIN

Άσκηση 35.1 · Events από κινητό για ένα προϊόν

Με το GIN index της θεωρίας, δείξε το plan με `EXPLAIN (COSTS OFF)` και μέτρησε τα events από `mobile` για το προϊόν 777.

ΛΥΣΗ

```
EXPLAIN (COSTS OFF)
SELECT count(*) FROM lab.events WHERE payload @> '{"device": "mobile", "product_id": 777}';

SELECT count(*) FROM lab.events WHERE payload @> '{"device": "mobile", "product_id": 777}';
```

ΑΠΟΤΕΛΕΣΜΑ

```
Aggregate
  ->  Bitmap Heap Scan on events
        Recheck Cond: (payload @> '{"device": "mobile", "product_id": 777}'::jsonb)
        ->  Bitmap Index Scan on events_payload_idx
              Index Cond: (payload @> '{"device": "mobile", "product_id": 777}'::jsonb)

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

Ένα `@>` με δύο κλειδιά ψάχνει γραμμές που περιέχουν και τα δύο ζεύγη. Το GIN συνδυάζει τις λίστες των δύο κομματιών και βρίσκει την τομή τους.

Άσκηση 35.2 · Αναζήτηση μέσα στην περιγραφή

Φτιάξε trigram GIN index στην περιγραφή του `lab.products` και δείξε με `EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)` ότι το χρησιμοποιεί η αναζήτηση `description ILIKE '%ξύλο οξιάς%'`.

ΛΥΣΗ

```
CREATE INDEX products_description_trgm_idx
  ON lab.products USING gin (description gin_trgm_ops);

EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)
SELECT count(*) FROM lab.products WHERE description ILIKE '%ξύλο οξιάς%';
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE INDEX

Aggregate (actual rows=1.00 loops=1)
  Buffers: shared hit=1152
  ->  Bitmap Heap Scan on products (actual rows=10108.00 loops=1)
        Recheck Cond: (description ~~* '%ξύλο οξιάς%'::text)
        Heap Blocks: exact=1116
        Buffers: shared hit=1152
        ->  Bitmap Index Scan on products_description_trgm_idx (actual rows=10108.00 loops=1)
              Index Cond: (description ~~* '%ξύλο οξιάς%'::text)
              Index Searches: 1
              Buffers: shared hit=36
Planning:
  Buffers: shared hit=30 read=5 dirtied=2
Planning Time: 0.578 ms
Execution Time: 13.367 ms
```

Το ίδιο είδος index δουλεύει για οποιαδήποτε στήλη κειμένου. Επειδή το μοτίβο ταιριάζει σε πολλά προϊόντα, το `Bitmap Heap Scan` διαβάζει πολλές σελίδες. Ένα index βοηθά περισσότερο όσο πιο σπάνιο είναι αυτό που ψάχνεις.

Άσκηση 35.3 · BRIN στα events

Φτιάξε BRIN index στο `occurred_at` του `lab.events` και B-tree index στην ίδια στήλη. Σύγκρινε τα μεγέθη τους.

ΛΥΣΗ

```
CREATE INDEX events_time_brin ON lab.events USING brin (occurred_at);
CREATE INDEX events_time_btree ON lab.events (occurred_at);

SELECT relname, pg_size_pretty(pg_relation_size(oid)) AS size
FROM pg_class
WHERE relname IN ('events_time_brin', 'events_time_btree')
ORDER BY pg_relation_size(oid);
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE INDEX

CREATE INDEX

      relname      | size
-------------------+-------
 events_time_brin  | 24 kB
 events_time_btree | 21 MB
(2 rows)
```

Τα events γράφτηκαν με τη σειρά του χρόνου, οπότε το BRIN είναι αποτελεσματικό και χιλιάδες φορές μικρότερο. Όταν υπάρχουν και τα δύο, ο planner διαλέγει συνήθως το B-tree για μικρά διαστήματα και το BRIN για μεγάλα.

Άσκηση 35.4 · Full text search με index

Πρόσθεσε στον `lab.products` stored generated column `search` με `to_tsvector('greek', name || ' ' || description)` και GIN index πάνω της. Δείξε το plan και μέτρησε τα προϊόντα που ταιριάζουν στην αναζήτηση `αθόρυβο γραφείο`.

ΛΥΣΗ

```
ALTER TABLE lab.products
  ADD COLUMN search tsvector
  GENERATED ALWAYS AS (to_tsvector('greek', name || ' ' || description)) STORED;
CREATE INDEX products_search_idx ON lab.products USING gin (search);

EXPLAIN (COSTS OFF)
SELECT count(*) FROM lab.products
WHERE search @@ websearch_to_tsquery('greek', 'αθόρυβο γραφείο');

SELECT count(*) FROM lab.products
WHERE search @@ websearch_to_tsquery('greek', 'αθόρυβο γραφείο');
```

ΑΠΟΤΕΛΕΣΜΑ

```
ALTER TABLE

CREATE INDEX

Aggregate
  ->  Bitmap Heap Scan on products
        Recheck Cond: (search @@ '''αθορυβ'' & ''γραφει'''::tsquery)
        ->  Bitmap Index Scan on products_search_idx
              Index Cond: (search @@ '''αθορυβ'' & ''γραφει'''::tsquery)

 count
-------
   904
(1 row)
```

Η στήλη πρέπει να είναι `STORED`, γιατί ένα index δεν μπορεί να στηθεί πάνω σε virtual generated column. Το `ALTER TABLE` ξαναγράφει τον πίνακα για να υπολογίσει τη νέα στήλη σε όλες τις γραμμές. Σε μεγάλο πίνακα αυτό είναι βαριά λειτουργία.

Άσκηση 35.5 · Κοντινότερα ονόματα με λάθη

Με το GiST index της θεωρίας, βρες τα πέντε προϊόντα με όνομα πιο κοντά στο ανορθόγραφο `Ακουστηκα Nova`, με την απόστασή τους. Δείξε και το plan.

ΛΥΣΗ

```
EXPLAIN (COSTS OFF)
SELECT name FROM lab.products ORDER BY name <-> 'Ακουστηκα Nova' LIMIT 5;

SELECT name, round((name <-> 'Ακουστηκα Nova')::numeric, 2) AS distance
FROM lab.products
ORDER BY name <-> 'Ακουστηκα Nova'
LIMIT 5;
```

ΑΠΟΤΕΛΕΣΜΑ

```
Limit
  ->  Index Scan using products_name_gist_idx on products
        Order By: (name <-> 'Ακουστηκα Nova'::text)

          name          | distance
------------------------+----------
 Ακουστικά Nova 9DF-977 |     0.58
 Ακουστικά Nova 2E0-273 |     0.58
 Ακουστικά Nova 93E-957 |     0.58
 Ακουστικά Nova 328-349 |     0.58
 Ακουστικά Nova 33D-300 |     0.58
(5 rows)
```

Δύο λάθη, το η αντί για ι και ο τόνος που λείπει, δεν εμπόδισαν την αναζήτηση. Τα trigrams του υπόλοιπου κειμένου είναι αρκετά για να βρεθούν τα ακουστικά Nova. Το plan είναι ένα `Index Scan` με `Order By`, όπως στη θεωρία.

ΚΕΦΑΛΑΙΟ 36

Τεχνικές για γρήγορα queries

Άσκηση 36.1 · Η επόμενη σελίδα

Η πρώτη σελίδα των παραγγελιών του πελάτη 777, από την πιο πρόσφατη, με πέντε παραγγελίες ανά σελίδα, τελειώνει στην παραγγελία με `created_at` και `id` που θα βρεις με το πρώτο query. Γράψε το query της δεύτερης σελίδας με keyset pagination.

ΛΥΣΗ

```
SELECT id, created_at
FROM lab.orders
WHERE customer_id = 777
ORDER BY created_at DESC, id DESC
LIMIT 5;

SELECT id, created_at
FROM lab.orders
WHERE customer_id = 777
  AND (created_at, id) < (SELECT created_at, id FROM lab.orders
                          WHERE customer_id = 777
                          ORDER BY created_at DESC, id DESC
                          OFFSET 4 LIMIT 1)
ORDER BY created_at DESC, id DESC
LIMIT 5;
```

ΑΠΟΤΕΛΕΣΜΑ

```
   id    |          created_at
---------+-------------------------------
 1969541 | 2026-05-31 11:57:26.260093+03
 1936268 | 2026-04-28 01:18:00.577505+03
 1917610 | 2026-04-09 09:31:01.185311+03
 1910564 | 2026-04-02 07:31:53.880285+03
 1892564 | 2026-03-15 05:36:54.261034+02
(5 rows)

   id    |          created_at
---------+-------------------------------
 1871039 | 2026-02-21 14:55:00.881456+02
 1855762 | 2026-02-06 06:20:40.011039+02
 1840112 | 2026-01-21 14:41:56.191803+02
 1810734 | 2025-12-23 03:20:47.100618+02
 1782802 | 2025-11-25 02:05:12.63263+02
(5 rows)
```

Σε φθίνουσα σειρά η σύγκριση γίνεται `<`. Σε πραγματική εφαρμογή οι τιμές της τελευταίας γραμμής θα ήταν παράμετροι που έστειλε ο client. Εδώ τις βρίσκει ένα subquery, ώστε η λύση να είναι ένα αυτόνομο query.

Άσκηση 36.2 · Σύγκριση σελιδοποίησης

Με `EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)`, σύγκρινε το query που φέρνει τις παραγγελίες 1.000.001 έως 1.000.010 κατά `id` με `OFFSET` και με keyset στο `id`. Εντόπισε τη διαφορά στα buffers.

ΛΥΣΗ

```
EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)
SELECT id, total FROM lab.orders ORDER BY id LIMIT 10 OFFSET 1000000;

EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)
SELECT id, total FROM lab.orders WHERE id > 1000000 ORDER BY id LIMIT 10;
```

ΑΠΟΤΕΛΕΣΜΑ

```
Limit (actual rows=10.00 loops=1)
  Buffers: shared hit=4336 read=6964 written=64
  ->  Index Scan using orders_pkey on orders (actual rows=1000010.00 loops=1)
        Index Searches: 1
        Buffers: shared hit=4336 read=6964 written=64
Planning Time: 0.011 ms
Execution Time: 79.683 ms

Limit (actual rows=10.00 loops=1)
  Buffers: shared hit=3 read=1
  ->  Index Scan using orders_pkey on orders (actual rows=10.00 loops=1)
        Index Cond: (id > 1000000)
        Index Searches: 1
        Buffers: shared hit=3 read=1
Planning Time: 0.039 ms
Execution Time: 0.021 ms
```

Με `OFFSET` η βάση διάβασε ένα εκατομμύριο εγγραφές του index και τις σελίδες του πίνακα που τους αντιστοιχούν. Με keyset διάβασε λίγες σελίδες. Όταν το κλειδί ταξινόμησης είναι μοναδικό, όπως εδώ, αρκεί μία στήλη στη σύγκριση.

Άσκηση 36.3 · Υπάρχει ακριβή παραγγελία

Γράψε με `EXISTS` query που επιστρέφει για τους πελάτες 1 έως 5 αν έχουν τουλάχιστον μία παραγγελία πάνω από 1490 ευρώ. Δείξε το plan με `EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)`.

ΛΥΣΗ

```
EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)
SELECT c.id,
       EXISTS (SELECT 1 FROM lab.orders AS o
               WHERE o.customer_id = c.id AND o.total > 1490) AS has_big_order
FROM lab.customers AS c
WHERE c.id <= 5;
```

ΑΠΟΤΕΛΕΣΜΑ

```
Index Only Scan using customers_pkey on customers c (actual rows=5.00 loops=1)
  Index Cond: (id <= 5)
  Heap Fetches: 0
  Index Searches: 1
  Buffers: shared hit=913 read=149 written=7
  SubPlan 1
    ->  Bitmap Heap Scan on orders o (actual rows=1.00 loops=5)
          Recheck Cond: (customer_id = c.id)
          Filter: (total > '1490'::numeric)
          Rows Removed by Filter: 206
          Heap Blocks: exact=983
          Buffers: shared hit=909 read=149 written=7
          ->  Bitmap Index Scan on orders_customer_created_idx (actual rows=2012.20 loops=5)
                Index Cond: (customer_id = c.id)
                Index Searches: 5
                Buffers: shared hit=31 read=22
Planning:
  Buffers: shared hit=5 read=6
Planning Time: 0.099 ms
Execution Time: 2.089 ms
```

Για κάθε πελάτη το subquery σταματά στην πρώτη παραγγελία που ταιριάζει. Οι πελάτες με μικρό `id` έχουν χιλιάδες παραγγελίες, οπότε η πρώτη ακριβή βρίσκεται γρήγορα. Το `count(*) > 0` θα έπρεπε να τις ελέγξει όλες.

Άσκηση 36.4 · Μια λίστα αντί για πολλά queries

Μια σελίδα δείχνει τους πελάτες 150010 έως 150014 και για τον καθένα το ποσό της πιο πρόσφατης παραγγελίας. Γράψε ένα query που δίνει και τους πέντε μαζί, με `= ANY` και `LATERAL`.

ΛΥΣΗ

```
SELECT c.id, c.first_name, last_o.total, last_o.created_at::date
FROM lab.customers AS c
LEFT JOIN LATERAL (SELECT o.total, o.created_at
                   FROM lab.orders AS o
                   WHERE o.customer_id = c.id
                   ORDER BY o.created_at DESC
                   LIMIT 1) AS last_o ON true
WHERE c.id = ANY (ARRAY[150010, 150011, 150012, 150013, 150014])
ORDER BY c.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
   id   | first_name |  total  | created_at
--------+------------+---------+------------
 150010 | Φωτεινή    |  372.28 | 2024-11-25
 150011 | Στέλιος    |  378.38 | 2026-04-04
 150012 | Νίκος      |  171.03 | 2025-12-27
 150013 | Αλέξανδρος |   14.13 | 2025-10-29
 150014 | Ζωή        | 1281.25 | 2024-11-25
(5 rows)
```

Ένα query αντικατέστησε έξι, ένα για τους πελάτες και ένα για τον καθένα. Το `LEFT JOIN LATERAL` κρατά και όσους δεν έχουν παραγγελίες.

Άσκηση 36.5 · Εκτίμηση ή ακρίβεια

Σύγκρινε την εκτίμηση του καταλόγου για το πλήθος των γραμμών του `lab.events` με το ακριβές `count(*)`. Δείξε πόσες σελίδες διάβασε το `count(*)`.

ΛΥΣΗ

```
SELECT reltuples::bigint AS estimated FROM pg_class WHERE oid = 'lab.events'::regclass;

EXPLAIN (ANALYZE, COSTS OFF, TIMING OFF)
SELECT count(*) FROM lab.events;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 estimated
-----------
   1000000
(1 row)

Aggregate (actual rows=1.00 loops=1)
  Buffers: shared read=13659 dirtied=13659 written=2554
  ->  Seq Scan on events (actual rows=1000000.00 loops=1)
        Buffers: shared read=13659 dirtied=13659 written=2554
Planning:
  Buffers: shared hit=16 read=3 dirtied=1
Planning Time: 0.128 ms
Execution Time: 60.792 ms
```

Η εκτίμηση είναι ακριβής εδώ, αφού ο πίνακας δεν έχει αλλάξει από το τελευταίο `VACUUM`. Το `count(*)` διάβασε το primary key ολόκληρο ή τον πίνακα, χιλιάδες σελίδες για έναν αριθμό που ο κατάλογος ήξερε ήδη.

ΚΕΦΑΛΑΙΟ 37

MVCC και isolation levels

Άσκηση 37.1 · Το isolation level ενός transaction

Μέσα σε transaction με `REPEATABLE READ`, εμφάνισε την τρέχουσα τιμή της ρύθμισης `transaction_isolation`. Μετά εμφάνισέ την ξανά εκτός transaction.

ΛΥΣΗ

```
BEGIN ISOLATION LEVEL REPEATABLE READ;
SHOW transaction_isolation;
COMMIT;
SHOW transaction_isolation;
```

ΑΠΟΤΕΛΕΣΜΑ

```
BEGIN

 transaction_isolation
-----------------------
 repeatable read
(1 row)

COMMIT

 transaction_isolation
-----------------------
 read committed
(1 row)
```

Το isolation level ορίζεται για κάθε transaction και επιστρέφει στην προεπιλογή μετά το τέλος του. Η προεπιλογή αλλάζει με τη ρύθμιση `default_transaction_isolation`.

Άσκηση 37.2 · Ένα snapshot για όλο το transaction

Σε transaction με `REPEATABLE READ`, εμφάνισε δύο φορές το `pg_current_snapshot()` με μια αλλαγή σε κάποια γραμμή ανάμεσα. Σύγκρινε τις δύο τιμές.

ΛΥΣΗ

```
BEGIN ISOLATION LEVEL REPEATABLE READ;
SELECT pg_current_snapshot() IS NOT NULL AS has_snapshot;
CREATE TEMP TABLE snap AS SELECT pg_current_snapshot()::text AS s;
UPDATE products SET stock = stock WHERE id = 3;
SELECT (SELECT s FROM snap) = pg_current_snapshot()::text AS same_snapshot;
ROLLBACK;
```

ΑΠΟΤΕΛΕΣΜΑ

```
BEGIN

 has_snapshot
--------------
 t
(1 row)

SELECT 1

UPDATE 1

 same_snapshot
---------------
 t
(1 row)

ROLLBACK
```

Το snapshot δεν άλλαξε, ούτε μετά από αλλαγή του ίδιου του transaction. Οι δικές του αλλαγές είναι πάντα ορατές σε αυτό, αλλά το snapshot που αποφασίζει τι βλέπει από τους άλλους μένει ίδιο. Ο `TEMP` πίνακας υπάρχει μόνο για τη σύνδεση και εδώ κρατά την πρώτη τιμή για σύγκριση.

Άσκηση 37.3 · Αγορά χωρίς υπερπώληση

Γράψε ένα `UPDATE` που μειώνει το απόθεμα του προϊόντος 26 κατά 2 μόνο αν υπάρχουν αρκετά τεμάχια. Να επιστρέφει το νέο απόθεμα. Δοκίμασέ το δύο φορές στο ίδιο transaction.

ΛΥΣΗ

```
UPDATE products SET stock = stock - 2 WHERE id = 26 AND stock >= 2 RETURNING stock;
UPDATE products SET stock = stock - 2 WHERE id = 26 AND stock >= 2 RETURNING stock;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 stock
-------
     1
(1 row)

UPDATE 1

 stock
-------
(0 rows)

UPDATE 0
```

Το προϊόν είχε τρία τεμάχια. Η πρώτη ενημέρωση πέτυχε και άφησε ένα. Η δεύτερη δεν βρήκε γραμμή που να ικανοποιεί τη συνθήκη και επέστρεψε `UPDATE 0`. Η εφαρμογή ελέγχει το πλήθος των γραμμών που άλλαξαν για να μάθει αν η αγορά έγινε. Με ταυτόχρονα transactions, το δεύτερο θα περίμενε το πρώτο και θα ξαναέλεγχε τη συνθήκη πάνω στη νέα έκδοση.

Άσκηση 37.4 · Ποιο transaction έγραψε τη γραμμή

Σε transaction, αύξησε κατά 1 το απόθεμα του προϊόντος 4. Εμφάνισε το `xmin` της γραμμής ως κείμενο και τον αριθμό του τρέχοντος transaction με `pg_current_xact_id()`. Έλεγξε αν είναι ίσα.

ΛΥΣΗ

```
UPDATE products SET stock = stock + 1 WHERE id = 4;
SELECT xmin::text = pg_current_xact_id()::text AS written_by_me
FROM products WHERE id = 4;
```

ΑΠΟΤΕΛΕΣΜΑ

```
UPDATE 1

 written_by_me
---------------
 t
(1 row)
```

Η νέα έκδοση της γραμμής γράφτηκε από το τρέχον transaction, οπότε το `xmin` της είναι ο αριθμός του. Η λύση συγκρίνει τα δύο ως κείμενο, αφού έχουν διαφορετικούς τύπους. Δεν τυπώνει τους αριθμούς, γιατί αλλάζουν σε κάθε εκτέλεση.

Άσκηση 37.5 · Σταθερός χρόνος μέσα στο transaction

Μέσα σε transaction, σύγκρινε το `now()` με το `transaction_timestamp()` και το `statement_timestamp()` με το `clock_timestamp()`, σε δύο διαδοχικές εντολές με μια μικρή καθυστέρηση ανάμεσα.

ΛΥΣΗ

```
SELECT now() = transaction_timestamp() AS now_is_transaction_start;
SELECT pg_sleep(0.01);
SELECT now() < statement_timestamp() AS statement_is_later,
       statement_timestamp() <= clock_timestamp() AS clock_is_latest;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 now_is_transaction_start
--------------------------
 t
(1 row)

 pg_sleep
----------

(1 row)

 statement_is_later | clock_is_latest
--------------------+-----------------
 t                  | t
(1 row)
```

Το `now()` είναι η στιγμή που ξεκίνησε το transaction και δεν αλλάζει μέσα σε αυτό. Το `statement_timestamp()` είναι η αρχή της τρέχουσας εντολής και το `clock_timestamp()` η πραγματική ώρα τη στιγμή της κλήσης. Όλες οι εγγραφές ενός transaction με `DEFAULT now()` παίρνουν την ίδια χρονοσφραγίδα.

ΚΕΦΑΛΑΙΟ 38

Locks, SKIP LOCKED και deadlocks

Άσκηση 38.1 · Διάβασε, αποφάσισε, γράψε

Σε transaction, κλείδωσε με `FOR UPDATE` το προϊόν 10 και διάβασε το απόθεμά του. Αν είναι τουλάχιστον 2, μείωσέ το κατά 2 με ένα `UPDATE` που επιστρέφει το νέο απόθεμα.

ΛΥΣΗ

```
SELECT id, stock FROM products WHERE id = 10 FOR UPDATE;
UPDATE products SET stock = stock - 2 WHERE id = 10 AND stock >= 2 RETURNING stock;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id | stock
----+-------
 10 |     6
(1 row)

 stock
-------
     4
(1 row)

UPDATE 1
```

Σε πραγματική εφαρμογή η απόφαση θα γινόταν στον κώδικα ανάμεσα στις δύο εντολές. Το lock εξασφαλίζει ότι κανείς δεν άλλαξε τη γραμμή στο μεταξύ. Εδώ το `AND stock >= 2` κάνει την ίδια απόφαση μέσα στη βάση.

Άσκηση 38.2 · Τα locks ενός transaction

Σε transaction, ενημέρωσε το προϊόν 11 και εμφάνισε από το `pg_locks` τα locks που κρατά η σύνδεσή σου στον πίνακα `products`, με το είδος και το όνομα του πίνακα.

ΛΥΣΗ

```
UPDATE products SET stock = stock WHERE id = 11;

SELECT locktype, relation::regclass AS relation, mode, granted
FROM pg_locks
WHERE pid = pg_backend_pid() AND relation = 'products'::regclass;
```

ΑΠΟΤΕΛΕΣΜΑ

```
UPDATE 1

 locktype | relation |       mode       | granted
----------+----------+------------------+---------
 relation | products | RowExclusiveLock | t
(1 row)
```

Ο πίνακας έχει `RowExclusiveLock`, που δεν εμποδίζει τις αναγνώσεις ούτε άλλα `UPDATE` σε άλλες γραμμές. Το lock της ίδιας της γραμμής δεν εμφανίζεται στο `pg_locks`. Γράφεται στο `xmax` της ίδιας της γραμμής. Γι' αυτό η PostgreSQL μπορεί να κλειδώνει εκατομμύρια γραμμές χωρίς να γεμίσει τη μνήμη.

Άσκηση 38.3 · Worker ουράς

Με τον πίνακα `jobs` της θεωρίας, γράψε σε ένα query την κίνηση ενός worker. Πάρε τις δύο παλαιότερες εργασίες που δεν έχουν γίνει με `SKIP LOCKED`, σημάδεψέ τες με `done_at` ίσο με `2026-06-30 11:00` και επέστρεψε τα `id` τους.

ΛΥΣΗ

```
WITH next_jobs AS (
  SELECT id FROM jobs
  WHERE done_at IS NULL
  ORDER BY created_at
  LIMIT 2
  FOR UPDATE SKIP LOCKED
)
UPDATE jobs AS j
SET done_at = '2026-06-30 11:00'
FROM next_jobs
WHERE j.id = next_jobs.id
RETURNING j.id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 id
----
  3
  4
(2 rows)

UPDATE 2
```

Το CTE κλειδώνει και το `UPDATE` σημαδεύει στο ίδιο query. Οι εργασίες 1 και 2 έγιναν στη θεωρία, οπότε ο worker πήρε την 3 και την 4.

Άσκηση 38.4 · Advisory lock ανά transaction

Σε transaction, πάρε το advisory lock με κλειδί 42 με `pg_advisory_xact_lock`. Εμφάνισε από το `pg_locks` το lock με `locktype` ίσο με `advisory` της σύνδεσής σου.

ΛΥΣΗ

```
SELECT pg_advisory_xact_lock(42);

SELECT locktype, objid, mode, granted
FROM pg_locks
WHERE pid = pg_backend_pid() AND locktype = 'advisory';
```

ΑΠΟΤΕΛΕΣΜΑ

```
 pg_advisory_xact_lock
-----------------------

(1 row)

 locktype | objid |     mode      | granted
----------+-------+---------------+---------
 advisory |    42 | ExclusiveLock | t
(1 row)
```

Το κλειδί εμφανίζεται στη στήλη `objid`. Το lock θα ελευθερωθεί μόνο του όταν τελειώσει το transaction, ακόμη κι αν ο κώδικας ξεχάσει να το ελευθερώσει ή πετάξει exception.

Άσκηση 38.5 · Όριο αναμονής

Σε transaction, όρισε `lock_timeout` 2 δευτερόλεπτα με `SET LOCAL` και εμφάνισε την τιμή του. Μετά το τέλος του transaction εμφάνισέ την ξανά.

ΛΥΣΗ

```
BEGIN;
SET LOCAL lock_timeout = '2s';
SHOW lock_timeout;
COMMIT;
SHOW lock_timeout;
```

ΑΠΟΤΕΛΕΣΜΑ

```
BEGIN

SET

 lock_timeout
--------------
 2s
(1 row)

COMMIT

 lock_timeout
--------------
 0
(1 row)
```

Το `SET LOCAL` ισχύει μόνο για το transaction. Είναι ο ασφαλής τρόπος να βάλεις όριο σε μια επικίνδυνη εντολή, όπως ένα `ALTER TABLE`, χωρίς να επηρεάσεις τη σύνδεση μετά. Το 0 σημαίνει χωρίς όριο.

ΚΕΦΑΛΑΙΟ 39

Functions, procedures και triggers

Άσκηση 39.1 · Ονοματεπώνυμο ως function

Φτιάξε `IMMUTABLE` SQL function `full_name(first text, last text)` που επιστρέφει όνομα και επώνυμο με ένα κενό ανάμεσα. Χρησιμοποίησέ τη για τους πελάτες 1 έως 3.

ΛΥΣΗ

```
CREATE FUNCTION full_name(first text, last text)
RETURNS text
LANGUAGE sql
IMMUTABLE
RETURN first || ' ' || last;

SELECT id, full_name(first_name, last_name) FROM customers WHERE id <= 3 ORDER BY id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE FUNCTION

 id |      full_name
----+---------------------
  1 | Μαρία Παπαδοπούλου
  2 | Γιώργος Παπαδόπουλος
  3 | Ελένη Νικολάου
(3 rows)
```

Η function διαβάζει μόνο τα ορίσματά της, οπότε το `IMMUTABLE` είναι ειλικρινές. Μπορεί να χρησιμοποιηθεί σε index, για παράδειγμα `CREATE INDEX ON customers (full_name(first_name, last_name))`.

Άσκηση 39.2 · Κορυφαία προϊόντα κατηγορίας

Φτιάξε function `top_products(p_category integer, p_n integer)` που επιστρέφει πίνακα με όνομα και έσοδα των `p_n` προϊόντων της κατηγορίας με τα περισσότερα έσοδα από παραγγελίες που δεν ακυρώθηκαν ούτε επιστράφηκαν. Κάλεσέ τη για την κατηγορία 9 και δύο προϊόντα.

ΛΥΣΗ

```
CREATE FUNCTION top_products(p_category integer, p_n integer)
RETURNS TABLE (name text, revenue numeric)
LANGUAGE sql
STABLE
BEGIN ATOMIC
  SELECT p.name, sum(oi.quantity * oi.unit_price)
  FROM order_items AS oi
  JOIN orders AS o ON o.id = oi.order_id
  JOIN products AS p ON p.id = oi.product_id
  WHERE p.category_id = p_category
    AND o.status NOT IN ('cancelled', 'refunded')
  GROUP BY p.id
  ORDER BY 2 DESC
  LIMIT p_n;
END;

SELECT * FROM top_products(9, 2);
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE FUNCTION

         name          | revenue
-----------------------+---------
 Οθόνη 27" 4K          | 3948.00
 USB-C docking station | 1112.00
(2 rows)
```

Οι παράμετροι έχουν πρόθεμα `p_` ώστε να μη συγχέονται με στήλες. Μια παράμετρος με όνομα `name` θα ήταν διφορούμενη μέσα στο query.

Άσκηση 39.3 · Email πάντα σε πεζά

Φτιάξε BEFORE trigger στον `customers` που μετατρέπει το email σε πεζά σε κάθε `INSERT` και `UPDATE`. Πρόσθεσε πελάτη με email `Nikos.Test@Example.GR` και επέστρεψε το email που αποθηκεύτηκε.

ΛΥΣΗ

```
CREATE FUNCTION lowercase_email()
RETURNS trigger
LANGUAGE plpgsql
AS $$
BEGIN
  NEW.email := lower(NEW.email);
  RETURN NEW;
END;
$$;

CREATE TRIGGER customers_lowercase_email
BEFORE INSERT OR UPDATE OF email ON customers
FOR EACH ROW
EXECUTE FUNCTION lowercase_email();

INSERT INTO customers (first_name, last_name, email, created_at)
VALUES ('Νίκος', 'Δοκιμής', 'Nikos.Test@Example.GR', '2026-06-30')
RETURNING email;
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE FUNCTION

CREATE TRIGGER

         email
-----------------------
 nikos.test@example.gr
(1 row)

INSERT 0 1
```

Το `RETURNING` δείχνει τη γραμμή όπως γράφτηκε, μετά το trigger. Το `UPDATE OF email` εκτελεί το trigger μόνο όταν ένα `UPDATE` αναφέρει τη στήλη email, οπότε οι υπόλοιπες ενημερώσεις δεν το πληρώνουν.

Άσκηση 39.4 · Απόθεμα που δεν γίνεται αρνητικό

Φτιάξε AFTER INSERT trigger στον `order_items` που μειώνει το απόθεμα του προϊόντος κατά την ποσότητα. Αν το απόθεμα δεν φτάνει, να σταματά με `RAISE EXCEPTION`. Δοκίμασέ το με δύο τεμάχια του προϊόντος 26 στην παραγγελία 162 και μετά με δέκα τεμάχια του προϊόντος 25 στην ίδια παραγγελία.

ΛΥΣΗ

```
CREATE FUNCTION reserve_stock()
RETURNS trigger
LANGUAGE plpgsql
AS $$
BEGIN
```

```
  UPDATE products SET stock = stock - NEW.quantity
  WHERE id = NEW.product_id AND stock >= NEW.quantity;
  IF NOT FOUND THEN
    RAISE EXCEPTION 'Δεν υπάρχει αρκετό απόθεμα για το προϊόν %', NEW.product_id;
  END IF;
  RETURN NULL;
END;
$$;

CREATE TRIGGER order_items_reserve_stock
AFTER INSERT ON order_items
FOR EACH ROW
EXECUTE FUNCTION reserve_stock();

INSERT INTO order_items VALUES (162, 26, 2, 459.00);
SELECT stock FROM products WHERE id = 26;
INSERT INTO order_items VALUES (162, 25, 10, 289.00);
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE FUNCTION

CREATE TRIGGER

INSERT 0 1

 stock
-------
     2
(1 row)

ERROR:  Δεν υπάρχει αρκετό απόθεμα για το προϊόν 25
CONTEXT:  PL/pgSQL function reserve_stock() line 6 at RAISE
```

Το `FOUND` είναι αληθές όταν η τελευταία εντολή επηρέασε τουλάχιστον μία γραμμή. Η δεύτερη εισαγωγή απέτυχε και η γραμμή της παραγγελίας δεν γράφτηκε, αφού το σφάλμα ακυρώνει ολόκληρη την εντολή. Ο έλεγχος μέσα στο `UPDATE` είναι ίδιος με την άσκηση 37.3 και αντέχει ταυτόχρονες παραγγελίες.

Άσκηση 39.5 · Μέτρηση με procedure

Φτιάξε procedure `deactivate_empty(INOUT p_count integer DEFAULT 0)` που απενεργοποιεί τα ενεργά προϊόντα με μηδενικό απόθεμα και επιστρέφει πόσα απενεργοποίησε. Κάλεσέ τη.

ΛΥΣΗ

```
CREATE PROCEDURE deactivate_empty(INOUT p_count integer DEFAULT 0)
LANGUAGE plpgsql
AS $$
BEGIN
  UPDATE products SET active = false WHERE active AND stock = 0;
  GET DIAGNOSTICS p_count = ROW_COUNT;
END;
$$;

CALL deactivate_empty();
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE PROCEDURE

 p_count
---------
       2
(1 row)
```

Μια procedure δεν έχει `RETURNS`. Επιστρέφει τιμές μέσω παραμέτρων `INOUT`, που το `CALL` εμφανίζει σαν γραμμή αποτελέσματος. Το `GET DIAGNOSTICS` διαβάζει πόσες γραμμές άλλαξε η τελευταία εντολή.

ΚΕΦΑΛΑΙΟ 40

Range types, exclusion constraints και partitioning

Άσκηση 40.1 · Πράξεις με διαστήματα

Για τα διαστήματα Μαΐου και Ιουνίου 2026 ως `daterange`, εμφάνισε αν επικαλύπτονται, αν είναι συνεχόμενα με τον τελεστή `-|-`, την ένωσή τους με `+` και τον αριθμό των ημερών της ένωσης.

ΛΥΣΗ

```
SELECT daterange('2026-05-01', '2026-06-01') && daterange('2026-06-01', '2026-07-01') AS overlap,
       daterange('2026-05-01', '2026-06-01') -|- daterange('2026-06-01', '2026-07-01') AS adjacent,
       daterange('2026-05-01', '2026-06-01') + daterange('2026-06-01', '2026-07-01') AS together,
       upper(daterange('2026-05-01', '2026-06-01') + daterange('2026-06-01', '2026-07-01'))
       - lower(daterange('2026-05-01', '2026-06-01') + daterange('2026-06-01', '2026-07-01')) AS days;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 overlap | adjacent |         together          | days
---------+----------+---------------------------+------
 f       | t        | [2026-05-01,2026-07-01) |   61
(1 row)
```

Τα δύο μισάνοιχτα διαστήματα δεν επικαλύπτονται αλλά είναι συνεχόμενα, οπότε η ένωση είναι ένα διάστημα 61 ημερών. Αν υπήρχε κενό ανάμεσά τους, το `+` θα αποτύγχανε, αφού το αποτέλεσμα δεν θα ήταν ένα διάστημα. Για τέτοιες περιπτώσεις υπάρχουν οι multiranges.

Άσκηση 40.2 · Επιτρεπτή προσφορά

Πρόσθεσε μια προσφορά 25% στο προϊόν 19 από τις 15 Ιουνίου 2026 ως την 1η Ιουλίου 2026. Μετά εμφάνισε όλες τις προσφορές του προϊόντος 19 κατά ημερομηνία έναρξης.

ΛΥΣΗ

```
INSERT INTO promotions (product_id, discount, during)
VALUES (19, 0.25, tstzrange('2026-06-15', '2026-07-01'));

SELECT discount, during FROM promotions WHERE product_id = 19 ORDER BY lower(during);
```

ΑΠΟΤΕΛΕΣΜΑ

```
INSERT 0 1

 discount |                       during
----------+-----------------------------------------------------
     0.20 | ["2025-11-20 00:00:00+02","2025-12-01 00:00:00+02")
     0.10 | ["2026-06-01 00:00:00+03","2026-06-15 00:00:00+03")
     0.25 | ["2026-06-15 00:00:00+03","2026-07-01 00:00:00+03")
(3 rows)
```

Η νέα προσφορά ξεκινά ακριβώς εκεί που τελειώνει η προηγούμενη. Με μισάνοιχτα διαστήματα δεν υπάρχει επικάλυψη και το exclusion constraint τη δέχεται.

Άσκηση 40.3 · Τιμή μετά την προσφορά

Για κάθε γραμμή παραγγελίας που αγοράστηκε ενώ ίσχυε προσφορά, εμφάνισε `order_id`, `product_id`, την τιμή μονάδας και την τιμή που θα είχε μετά την έκπτωση, στρογγυλεμένη σε δύο δεκαδικά.

ΛΥΣΗ

```
SELECT oi.order_id, oi.product_id, oi.unit_price,
       round(oi.unit_price * (1 - promo.discount), 2) AS discounted
FROM order_items AS oi
JOIN orders AS o ON o.id = oi.order_id
JOIN promotions AS promo ON promo.product_id = oi.product_id
                        AND promo.during @> o.created_at
ORDER BY oi.order_id;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 order_id | product_id | unit_price | discounted
----------+------------+------------+------------
       58 |         19 |     199.00 |     159.20
      114 |         12 |    2199.00 |    1869.15
      130 |         12 |    2199.00 |    1869.15
      143 |         19 |     199.00 |     179.10
(4 rows)
```

Τα δεδομένα του καταστήματος γράφτηκαν χωρίς προσφορές, οπότε η τιμή μονάδας είναι η κανονική. Σε πραγματικό σύστημα η έκπτωση θα εφαρμοζόταν τη στιγμή της παραγγελίας και θα αποθηκευόταν στην `unit_price`, όπως εξήγησε το κεφάλαιο 18.

Άσκηση 40.4 · Pruning σε δύο partitions

Δείξε με `EXPLAIN (COSTS OFF)` το plan για τα events του `events_log` από τις 15 Δεκεμβρίου 2025 ως τις 15 Ιανουαρίου 2026.

ΛΥΣΗ

```
EXPLAIN (COSTS OFF)
SELECT count(*) FROM events_log
WHERE occurred_at >= '2025-12-15' AND occurred_at < '2026-01-15';
```

ΑΠΟΤΕΛΕΣΜΑ

```
Aggregate
  ->  Append
        ->  Seq Scan on events_log_2025h2 events_log_1
              Filter: ((occurred_at >= '2025-12-15 00:00:00+02'::timestamp with time
zone) AND (occurred_at < '2026-01-15 00:00:00+02'::timestamp with time zone))
        ->  Seq Scan on events_log_2026h1 events_log_2
              Filter: ((occurred_at >= '2025-12-15 00:00:00+02'::timestamp with time
zone) AND (occurred_at < '2026-01-15 00:00:00+02'::timestamp with time zone))
```

Το διάστημα περνά από το ένα εξάμηνο στο άλλο, οπότε το plan διαβάζει δύο partitions και τα ενώνει με `Append`. Το partition του πρώτου εξαμήνου του 2025 αποσπάστηκε στη θεωρία και δεν ανήκει πια στον πίνακα.

Άσκηση 40.5 · Partition για τα υπόλοιπα

Πρόσθεσε `DEFAULT` partition στον `events_log` και ξαναδοκίμασε την εισαγωγή του event της 15ης Ιουλίου 2026. Εμφάνισε σε ποιο partition κατέληξε.

ΛΥΣΗ

```
CREATE TABLE events_log_default PARTITION OF events_log DEFAULT;

INSERT INTO events_log VALUES (999999, 1, 'view', '2026-07-15')
RETURNING tableoid::regclass AS partition;
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE TABLE

     partition
--------------------
 events_log_default
(1 row)

INSERT 0 1
```

Το default partition δέχεται ό,τι δεν ταιριάζει αλλού. Έχει ένα κόστος. Όταν αργότερα φτιάξεις partition για το δεύτερο εξάμηνο του 2026, η βάση πρέπει να ελέγξει ότι το default δεν περιέχει ήδη γραμμές αυτού του διαστήματος.

ΚΕΦΑΛΑΙΟ 41

Ρόλοι, δικαιώματα και Row Level Security

Άσκηση 41.1 · Ρόλος μόνο για ανάγνωση

Φτιάξε ρόλο `book_viewer` χωρίς login με δικαίωμα ανάγνωσης στον `cities`. Υποδύσου τον με `SET ROLE` και δοκίμασε ένα `SELECT` και ένα `UPDATE`.

ΛΥΣΗ

```
CREATE ROLE book_viewer NOLOGIN;
GRANT SELECT ON cities TO book_viewer;

SET ROLE book_viewer;
SELECT count(*) FROM cities;
UPDATE cities SET population = population + 1 WHERE id = 1;
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE ROLE

GRANT

SET

 count
-------
    11
(1 row)

ERROR:  permission denied for table cities
```

Το `SELECT` πέτυχε και το `UPDATE` απορρίφθηκε. Η λύση εκτελείται σε transaction που αναιρείται, οπότε το `SET ROLE` και ο ρόλος χάνονται μαζί. Σε πραγματική σύνδεση θα έγραφες `RESET ROLE`.

Άσκηση 41.2 · Ποιος έχει τι

Εμφάνισε από το `information_schema.role_table_grants` όλα τα δικαιώματα του `book_reporting` σε πίνακες, ένα ανά γραμμή, ταξινομημένα.

ΛΥΣΗ

```
SELECT table_name, privilege_type
FROM information_schema.role_table_grants
WHERE grantee = 'book_reporting'
ORDER BY table_name, privilege_type;
```

ΑΠΟΤΕΛΕΣΜΑ

```
 table_name  | privilege_type
-------------+----------------
 categories  | SELECT
 order_items | SELECT
 orders      | SELECT
 products    | SELECT
(4 rows)
```

Τα δικαιώματα ανά στήλη στον `customers` δεν εμφανίζονται εδώ, αφού ο ρόλος δεν έχει δικαίωμα σε ολόκληρο τον πίνακα. Βρίσκονται στο `information_schema.column_privileges`.

Άσκηση 41.3 · Κριτικές μόνο του εαυτού σου

Ενεργοποίησε RLS στον `reviews` με policy για τον `book_app` που επιτρέπει να βλέπει όλες τις κριτικές αλλά να αλλάζει μόνο όσες έγραψε ο πελάτης της ρύθμισης `app.customer_id`. Δώσε στον `book_app` `SELECT` και `UPDATE` στον πίνακα. Ως πελάτης 1, άλλαξε τον τίτλο όλων των κριτικών για το προϊόν 19 και δες πόσες άλλαξαν.

ΛΥΣΗ

```
ALTER TABLE reviews ENABLE ROW LEVEL SECURITY;
GRANT SELECT, UPDATE ON reviews TO book_app;

CREATE POLICY reviews_read ON reviews FOR SELECT TO book_app USING (true);
CREATE POLICY reviews_update_own ON reviews FOR UPDATE TO book_app
  USING (customer_id = NULLIF(current_setting('app.customer_id', true), '')::integer)
  WITH CHECK (customer_id = NULLIF(current_setting('app.customer_id', true), '')::integer);

SET LOCAL ROLE book_app;
SET LOCAL app.customer_id = '1';
SELECT count(*) AS visible FROM reviews WHERE product_id = 19;
UPDATE reviews SET title = title || ' (ενημέρωση)' WHERE product_id = 19;
```

ΑΠΟΤΕΛΕΣΜΑ

```
ALTER TABLE

GRANT

CREATE POLICY

CREATE POLICY

SET

SET

 visible
---------
       4
(1 row)

UPDATE 1
```

Ο `book_app` βλέπει και τις τέσσερις κριτικές του προϊόντος, αλλά το `UPDATE` άλλαξε μόνο τη μία του πελάτη 1. Κάθε είδος εντολής μπορεί να έχει δική του policy. Όταν ένας ρόλος έχει πολλές policies για το ίδιο είδος εντολής, αρκεί να ισχύει μία.

Άσκηση 41.4 · Χωρίς πελάτη, τίποτα

Ως `book_app` και χωρίς να ορίσεις `app.customer_id`, μέτρησε τις παραγγελίες που βλέπεις. Μετά όρισε τον πελάτη 6 και μέτρησε ξανά.

ΛΥΣΗ

```
SET LOCAL ROLE book_app;
SELECT count(*) AS without_customer FROM orders;
SET LOCAL app.customer_id = '6';
SELECT count(*) AS as_customer_6 FROM orders;
```

ΑΠΟΤΕΛΕΣΜΑ

```
SET

 without_customer
------------------
                 0
(1 row)

SET

 as_customer_6
---------------
            10
(1 row)
```

Χωρίς ρύθμιση η policy συγκρίνει με `NULL` και δεν βλέπεις τίποτα. Αυτό είναι καλύτερο από την εναλλακτική, όπου ένα λάθος στην εφαρμογή θα έδειχνε όλες τις παραγγελίες όλων των πελατών.

Άσκηση 41.5 · View με τα δικαιώματα του αναγνώστη

Φτιάξε view `my_orders` πάνω στον `orders` με `security_invoker = true` και στήλες `id`, `status` και `created_at`. Δώσε `SELECT` στον `book_app`. Ως `book_app` και πελάτης 13, μέτρησε τις γραμμές του view.

ΛΥΣΗ

```
CREATE VIEW my_orders WITH (security_invoker = true) AS
SELECT id, status, created_at FROM orders;
GRANT SELECT ON my_orders TO book_app;

SET LOCAL ROLE book_app;
SET LOCAL app.customer_id = '13';
SELECT count(*) FROM my_orders;
```

ΑΠΟΤΕΛΕΣΜΑ

```
CREATE VIEW
GRANT
SET
SET
 count
-------
     5
(1 row)
```

Επειδή το view εκτελείται με τα δικαιώματα του `book_app`, ισχύει η policy του `orders` και το view δείχνει μόνο τις πέντε παραγγελίες του πελάτη 13. Χωρίς `security_invoker`, το view θα εκτελούνταν ως ο ιδιοκτήτης του, που παρακάμπτει το RLS. Θα έδειχνε όλες τις παραγγελίες.