

# 1η ΕΡΓΑΣΙΑ στη ΣΧΕΔΙΑΣΗ ΒΔ

## 2023-24

### 1ο ερώτημα - Ευρετήριο με B+ δένδρο

#### A ερώτημα:

##### 1) Αρχείο Σωρού:

Για να υπολογίσετε το μέγεθος του αρχείου ως προς το ID, πρέπει να λάβετε υπόψη τα μεγέθη όλων των στοιχείων του αρχείου, συμπεριλαμβανομένων των εγγραφών, του ευρετηρίου και των δεικτών.

Ας υπολογίσουμε τα μεγέθη των διαφόρων στοιχείων:

- Μέγεθος της εγγραφής στο αρχείο: 200 bytes.
- Αριθμός εγγραφών: 15,000,000.

Άρα το συνολικό μέγεθος των εγγραφών στο αρχείο είναι:

$200 \text{ bytes/εγγραφή} * 15,000,000 \text{ εγγραφές} = 3,000,000,000 \text{ bytes.}$

- Μέγεθος του block του αρχείου: 1024 bytes.
- Μέγεθος του δείκτη προς τις εγγραφές στο αρχείο: 12 bytes.
- Μέγεθος του δείκτη προς block του ευρετηρίου: 16 bytes.
- Μέγεθος του δείκτη προς επόμενο φύλλο του ευρετηρίου: 16 bytes.

Κάθε φύλλο του B\* δέντρου αναφέρεται σε ένα block του αρχείου και περιλαμβάνει δείκτες προς τις εγγραφές στο αρχείο και το επόμενο φύλλο.

Έστω ότι έχουμε N φύλλα στο δέντρο. Σε κάθε φύλλο, έχουμε:

- 12 bytes για τον δείκτη προς τις εγγραφές.
- 16 bytes για τον δείκτη προς block του αρχείου.
- 16 bytes για τον δείκτη προς το επόμενο φύλλο.
-

Άρα το μέγεθος κάθε φύλλου είναι: 12 bytes + 16 bytes + 16 bytes  
= 44 bytes.

Συνολικό μέγεθος φύλλων: 44 bytes/φύλλο \* N φύλλα = 44N bytes.

Το συνολικό μέγεθος του αρχείου είναι η συνολική αθροιστική ποσότητα των μεγεθών αυτών:

Συνολικό μέγεθος αρχείου = Μέγεθος εγγραφών + Μέγεθος φύλλων

Συνολικό μέγεθος αρχείου = 3,000,000,000 bytes + 44N bytes

Για να υπολογίσουμε το N, πρέπει να ξέρουμε πόσα φύλλα χρειάζονται για να αποθηκεύσουν όλες τις εγγραφές.

Καθώς τα φύλλα είναι όσο γίνεται πλήρη, κάθε φύλλο μπορεί να αποθηκεύσει τόσες εγγραφές όσες ταιριάζουν σε ένα block του αρχείου :

$1024 \text{ bytes} / 200 \text{ bytes/εγγραφή} = 5$

Οπότε κάθε φύλλο μπορεί να αποθηκεύσει 5 εγγραφές.

Άρα, ο αριθμός των φύλλων είναι:

$N = \text{Σύνολο εγγραφών} / \text{Εγγραφές ανά φύλλο}$

$= 15,000,000 / 5$

$= 3,000,000 \text{ φύλλα.}$

Συνολικό μέγεθος αρχείου = 3,000,000,000 bytes + 44 \* 3,000,000 bytes

$= 3,000,000,000 \text{ bytes} + 120,000,000 \text{ bytes}$

$= 3,120,000,000 \text{ bytes.}$

Άρα το μέγεθος του αρχείου ως προς το ID είναι 3,120,000,000 bytes, ή 3.12 GB (gigabytes).

## 2) Αρχείο Κατακερματισμού

Έχουμε αριθμό εγγράφων σε σωρό 15.000.000 και το μέγεθος της εγγραφής είναι 200 bytes άρα το συνολικό μέγεθος των εγγραφών είναι  $3.000.000.000/1024=2.929.687.5$  άρα αυτός είναι ο αριθμός των blocks άρα Έστω ότι διατηρούμε κάθε κάδο γεμάτο κατά 80% άρα ένα αρχείο με πλήθος block=B, τότε χρειάζεται  $1.25*B$  block δίσκου

Άρα χρειάζεται  $1,25*B$  block= 3.662.109,375 bytes

## 3) και 4)

Έχουμε:

- Μέγεθος block B = 1024 bytes
- Πλήθος Εγγράφων = 15.000.000
- Μέγεθος Εγγραφής = 200 bytes
- Μέγεθος Πεδίου Αναζήτησης(τιμή) V = 20 bytes
- Μέγεθος Δείκτη Εγγραφής Pr = 12 bytes
- Μέγεθος Δείκτη Μπλοκ P = 16 bytes
- Μέγεθος Δείκτη Επόμενου Φύλλου P = 16 bytes

Δείκτης block = Δείκτη επόμενου φύλλου, διότι σε ένα B\* δέντρο κάθε κόμβος είναι σαν block

Τάξη κόμβου = πλήθος δεικτών(προς κόμβους για εσωτερικούς κόμβους, προς έγγραφο για φύλλα)

Τάξη κόμβου - 1 = πλήθος τιμών κόμβου

Ένας κόμβος πρέπει να είναι το πολύ ίσος με το μέγεθος ενός block

Υπολογισμός τάξης εσωτερικών κόμβων:

$$p * P + (p - 1) * V \leq B$$

$$p \leq (B + V) / (P + V)$$

$$p \leq 29$$

Άρα οι εσωτερικοί κόμβοι είναι τάξης 29

Όμως, γνωρίζοντας από την εκφώνηση ότι μόνο το επίπεδο φύλλων είναι γεμάτο θα υποθέσουμε ότι τα υπόλοιπα επίπεδα είναι στο ελάχιστο δυνατό γεμάτα, δηλαδή:

Τα B\* δέντρα έχουν ελάχιστη χωρητικότητα κόμβων  $\frac{2}{3}$

$$p' = \frac{2}{3} * p = 19,3$$

Άρα οι εσωτερικοί κόμβοι είναι της τάξης 19.

- Έχουν 19 δείκτες προς του κόμβους του επόμενου επιπέδου
- Έχουν 18 τιμές που μπορούν να αποθηκεύσουν

$$p' = 19$$

Υπολογισμός τάξης κόμβων-φύλλων :

$$p_{leaf} * (P_r + V) + P \leq B$$

$$p_{leaf} \leq (B - P) / (P_r + V)$$

$$p_{leaf} \leq 31.5$$

Άρα τα φύλλα είναι τάξης  $p_{leaf} = 31$

- Έχουν 31 δείκτες προς έγγραφα.

$$\text{Μέγεθος εσωτερικών κόμβων και ρίζας} = p' * P + (p' - 1) * V$$

$$= 664 \text{ bytes}$$

$$\text{Μέγεθος κόμβων-φύλλων} = p_{leaf} * (P_r + V) + P$$

$$= 1008 \text{ bytes}$$

Το πλήθος των κόμβων ανά επίπεδο θα στρογγυλοποιείται προς τα πάνω διότι δεν μπορούμε να έχουμε μισό κόμβο

$\text{πλήθος\_φύλλων} = \text{πλήθος\_δεικτών\_εγγράφων\_επιπέδου\_φύλλων}$ .

$$15.000.000 = \text{πλήθος\_φύλλων} * \text{rleaf}$$

$$\text{πλήθος\_φύλλων} = 483.871$$

Άρα το τελευταίο επίπεδο( επίπεδο φύλλων) έχει 483.871 κόμβους

Από εδώ και πέρα μετράμε τους τα εσωτερικά επίπεδα.

Θα ξέρουμε ότι φτάσαμε στη ρίζα επειδή η ρίζα πρέπει να είναι ένας κόμβος μόνος του.

Για να έχουμε 483.871 κόμβους-φύλλα, στο αμέσως πάνω επίπεδο πρέπει να έχουμε συνολικά 483.871 δείκτες αλλά ξέρουμε ότι για κάθε δείκτη στο επάνω επίπεδο έχουμε μια λιγότερη τιμή ανά κόμβο

Άρα,

$$483.871 = \text{πλήθος\_κόμβων} * \text{r'}$$

$$\text{πλήθος\_κόμβων} = 25.467$$

Το επίπεδο έχει 25.467 κόμβους.

Για να έχουμε 25.467 κόμβους, στο αμέσως πάνω επίπεδο πρέπει να έχουμε συνολικά 25.467 δείκτες αλλά ξέρουμε ότι για κάθε δείκτη στο επάνω επίπεδο έχουμε μια λιγότερη τιμή ανά κόμβο

Άρα,

$$25.467 = \text{πλήθος\_κόμβων} * \text{r'}$$

$$\text{πλήθος\_κόμβων} = 1.341$$

Το επίπεδο έχει 1.341 κόμβους.

Για να έχουμε 1.341 κόμβους, στο αμέσως πάνω επίπεδο πρέπει να έχουμε συνολικά 1.341 δείκτες αλλά ξέρουμε ότι για κάθε δείκτη στο επάνω επίπεδο έχουμε μια λιγότερη τιμή ανά κόμβο

Άρα,

$$1.341 = \text{πλήθος\_κόμβων} * p'$$

$$\text{πλήθος\_κόμβων} = 71$$

Το επίπεδο έχει 71 κόμβους.

Για να έχουμε 71 κόμβους, στο αμέσως πάνω επίπεδο πρέπει να έχουμε συνολικά 71 δείκτες αλλά ξέρουμε ότι για κάθε δείκτη στο επάνω επίπεδο έχουμε μια λιγότερη τιμή ανά κόμβο

Άρα,

$$71 = \text{πλήθος\_κόμβων} * p'$$

$$\text{πλήθος\_κόμβων} = 4$$

Το επίπεδο έχει 4 κόμβους.

Για να έχουμε 4 κόμβους, στο αμέσως πάνω επίπεδο πρέπει να έχουμε συνολικά 4 δείκτες αλλά ξέρουμε ότι για κάθε δείκτη στο επάνω επίπεδο έχουμε μια λιγότερη τιμή ανά κόμβο

Άρα,

$$4 = \text{πλήθος\_κόμβων} * p'$$

$$\text{πλήθος\_κόμβων} = 1$$

Το επίπεδο έχει 1 κόμβο, άρα είναι η ρίζα.

Άρα, έχουμε 6 επίπεδα

Επίπεδα:

- Επίπεδο 0 ( ρίζα):
  - 1 κόμβος
  - μέγεθος = 664 bytes
- Επίπεδο 1
  - 4 κόμβοι
  - μέγεθος =  $4 * 664$  βυτες = 2.656 bytes
- Επίπεδο 2
  - 71 κόμβοι
  - μέγεθος =  $71 * 664$  βυτες = 47.144 bytes
- Επίπεδο 3
  - 1.341 κόμβοι
  - μέγεθος =  $1.341 * 664$  βυτες = 890.424 bytes
- Επίπεδο 4
  - 25.467 κόμβοι
  - μέγεθος =  $25.467 * 664$  βυτες = 16.910.088 bytes
- Επίπεδο 5 ( επίπεδο φύλλων )
  - 483.871 κόμβοι
  - μέγεθος =  $483.871 * 1008$  βυτες = 487.741.968 bytes

Μέγεθος Ευρετηρίου = μέγεθος\_0 + μέγεθος\_1 + μέγεθος\_2 + μέγεθος\_3 + μέγεθος\_4 + μέγεθος\_5 = 505.592.944 bytes

Άρα το μέγεθος του ευρετηρίου είναι 505.592.944 bytes.

5)

Έχουμε:

- Μέγεθος block B = 1024 bytes
- Πλήθος Εγγράφων = 15.000.000
- Μέγεθος Εγγραφής = 200 bytes
- Μέγεθος Πεδίου Αναζήτησης(τιμή) V = 20 bytes
- Μέγεθος Δείκτη Εγγραφής Pr = 12 bytes
- Μέγεθος Δείκτη Μπλοκ P = 16 bytes
- Μέγεθος Δείκτη Επόμενου Φύλλου P = 16 bytes

Δείκτης block = Δείκτη επόμενου φύλλου, διότι σε ένα B+ δέντρο κάθε κόμβος είναι σαν block

Τάξη κόμβου = πλήθος δεικτών(προς κόμβους για εσωτερικούς κόμβους, προς έγγραφο για φύλλα)

Τάξη κόμβου - 1 = πλήθος τιμών κόμβου

Ένας κόμβος πρέπει να είναι το πολύ ίσος με το μέγεθος ενός block

Υπολογισμός τάξης εσωτερικών κόμβων:

$$p * P + (p - 1) * V \leq B$$

$$p \leq (B + V) / (P + V)$$

$$p \leq 29$$

Άρα οι εσωτερικοί κόμβοι είναι τάξης 29

Όμως, γνωρίζοντας από την εκφώνηση ότι μόνο το επίπεδο φύλλων είναι γεμάτο θα υποθέσουμε ότι τα υπόλοιπα επίπεδα είναι στο ελάχιστο δυνατό γεμάτα, δηλαδή:

Τα B\* δέντρα έχουν ελάχιστη χωρητικότητα κόμβων  $\frac{1}{2}$

$$p' = \frac{1}{2} * p = 14,5$$

Άρα οι εσωτερικοί κόμβοι είναι της τάξης 14.

- Έχουν 14 δείκτες προς του κόμβους του επόμενου επιπέδου
- Έχουν 13 τιμές που μπορούν να αποθηκεύσουν

$$p' = 14$$

Υπολογισμός τάξης κόμβων-φύλλων :

$$p_{leaf} * (P + V) + P \leq B$$



$$p_{leaf} \leq (B - P) / (P_r + V)$$

$$p_{leaf} \leq 31.5$$

Άρα τα φύλλα είναι τάξης  $p_{leaf} = 31$

- Έχουν 31 δείκτες προς έγγραφα.

$$\begin{aligned} \text{Μέγεθος εσωτερικών κόμβων και ρίζας} &= p' * P + (p' - 1) * V \\ &= 484 \text{ bytes} \end{aligned}$$

$$\begin{aligned} \text{Μέγεθος κόμβων-φύλλων} &= p_{leaf} * (P_r + V) + P \\ &= 1008 \text{ bytes} \end{aligned}$$

Το πλήθος των κόμβων ανά επίπεδο θα στρογγυλοποιείται προς τα πάνω διότι δεν μπορούμε να έχουμε μισό κόμβο

$\text{πλήθος\_φύλλων} = \text{πλήθος\_δεικτών\_εγγράφων\_επιπέδου\_φύλλων}$ .

$$15.000.000 = \text{πλήθος\_φύλλων} * p_{leaf}$$

$$\text{πλήθος\_φύλλων} = 483.871$$

Άρα το τελευταίο επίπεδο( επίπεδο φύλλων) έχει 483.871 κόμβους

Από εδώ και πέρα μετράμε τους τα εσωτερικά επίπεδα.

Θα ξέρουμε ότι φτάσαμε στη ρίζα επειδή η ρίζα πρέπει να είναι ένας κόμβος μόνος του.

Για να έχουμε 483.871 κόμβους-φύλλα, στο αμέσως πάνω επίπεδο πρέπει να έχουμε συνολικά 483.871 δείκτες αλλά ξέρουμε ότι για κάθε δείκτη στο επάνω επίπεδο έχουμε μια λιγότερη τιμή ανά κόμβο

Άρα,

$$483.871 = \text{πλήθος\_κόμβων} * p'$$

$$\text{πλήθος\_κόμβων} = 34.563$$

Το επίπεδο έχει 34.563 κόμβους.

Για να έχουμε 34.563 κόμβους, στο αμέσως πάνω επίπεδο πρέπει να έχουμε συνολικά 34.563 δείκτες αλλά ξέρουμε ότι για κάθε δείκτη στο επάνω επίπεδο έχουμε μια λιγότερη τιμή ανά κόμβο

Άρα,

$$34.563 = \text{πλήθος\_κόμβων} * p'$$

$$\text{πλήθος\_κόμβων} = 2.469$$

Το επίπεδο έχει 2.469 κόμβους.

Για να έχουμε 2.469 κόμβους, στο αμέσως πάνω επίπεδο πρέπει να έχουμε συνολικά 2.469 δείκτες αλλά ξέρουμε ότι για κάθε δείκτη στο επάνω επίπεδο έχουμε μια λιγότερη τιμή ανά κόμβο

Άρα,

$$2.469 = \text{πλήθος\_κόμβων} * p'$$

$$\text{πλήθος\_κόμβων} = 177$$

Το επίπεδο έχει 177 κόμβους.

Για να έχουμε 177 κόμβους, στο αμέσως πάνω επίπεδο πρέπει να έχουμε συνολικά 177 δείκτες αλλά ξέρουμε ότι για κάθε δείκτη στο επάνω επίπεδο έχουμε μια λιγότερη τιμή ανά κόμβο

Άρα,

$$177 = \text{πλήθος\_κόμβων} * p'$$

$$\text{πλήθος\_κόμβων} = 13$$

Το επίπεδο έχει 13 κόμβους.

Για να έχουμε 13 κόμβους, στο αμέσως πάνω επίπεδο πρέπει να έχουμε συνολικά 13 δείκτες αλλά ξέρουμε ότι για κάθε δείκτη στο επάνω επίπεδο έχουμε μια λιγότερη τιμή ανά κόμβο

Άρα,

$$13 = \text{πλήθος\_κόμβων} * p'$$

$$\text{πλήθος\_κόμβων} = 1$$

Το επίπεδο έχει 1 κόμβο, άρα είναι η ρίζα.

Άρα, έχουμε 6 επίπεδα

Επίπεδα:

- Επίπεδο 0 ( ρίζα):
  - 1 κόμβος
  - μέγεθος = 484 bytes
- Επίπεδο 1
  - 13 κόμβοι
  - μέγεθος =  $13 * 484 \text{ bytes} = 6.292 \text{ bytes}$
- Επίπεδο 2
  - 177 κόμβοι
  - μέγεθος =  $177 * 484 \text{ bytes} = 85.668 \text{ bytes}$
- Επίπεδο 3
  - 2.469 κόμβοι
  - μέγεθος =  $2.469 * 484 \text{ bytes} = 1.194.996 \text{ bytes}$
- Επίπεδο 4
  - 34.563 κόμβοι
  - μέγεθος =  $34.563 * 484 \text{ bytes} = 16.728.492 \text{ bytes}$
- Επίπεδο 5 ( επίπεδο φύλλων )
  - 483.871 κόμβοι
  - μέγεθος =  $483.871 * 1008 \text{ bytes} = 487.741.968 \text{ bytes}$

$$\begin{aligned} \text{Μέγεθος Ευρετηρίου} &= \text{μέγεθος}_0 + \text{μέγεθος}_1 + \text{μέγεθος}_2 + \\ &+ \text{μέγεθος}_3 + \text{μέγεθος}_4 + \text{μέγεθος}_5 = 505.592.944 \text{ bytes} \end{aligned}$$

Άρα το μέγεθος του ευρετηρίου είναι 505.757.900 bytes.

6) Για την αναζήτηση στο δέντρο θα κάνει όσο είναι το ύψος του δέντρου  $O(h)=5$

Διακρίνουμε δύο περιπτώσεις :

1)Αν το πεδίο ισότητας είναι πρωτεύον κλειδί ->τότε έχουμε κόστος αναζήτησης + πλήθος μπλοκ απάντησης ισότητας $\leq$   $h+20=25$

2)Αν το πεδίο ισότητας δεν είναι πρωτεύον κλειδί ,διακρίνουμε δύο περιπτώσεις:

Αν το αρχείο είναι ταξινομημένο ως προς το πεδίο ή όχι:

α)Μη ταξινομημένο ->τότε έχουμε αναζήτηση στο δέντρο+μπλοκς των φύλων=  $5+483.871$

β)Ταξινομημένο -> κοστος αναζήτησης στο δέντρο + πλήθος μπλοκς απαντησης της ισοτητας / εγγραφές ανα μπλοκ αρχείου  $=5+20/5=9$

## **1 B Ερώτημα:**

Ο κώδικας δημιουργεί τρεις συναρτήσεις (get\_age\_group, get\_income\_level, fix\_status) και τρεις πίνακες (CUSTOMERS, PRODUCTS, ORDERS). Ας περιγράψουμε τις συναρτήσεις και τους πίνακες.

### **Συνάρτηση `get_age_group`**

Η συνάρτηση χρησιμοποιείται για τον υπολογισμό της ηλικιακής ομάδας ενός πελάτη με βάση την ημερομηνία γέννησής του. Η ηλικιακή ομάδα καθορίζεται σε πέντε κατηγορίες: 'under 40', '40-49', '50-59', '60-69', 'above 70', ή 'NON RECORDABLE' για μη έγκυρες ημερομηνίες.

### **Συνάρτηση `get_income_level`**

Αυτή η συνάρτηση μετατρέπει ένα εύρος εισοδήματος (σε μορφή συμβολοσειράς τύπου '1111 - 2222') σε τρεις κατηγορίες εισοδήματος: 'low', 'medium', ή 'high'. Ελέγχει τις τιμές υψηλότερου εισοδήματος και καθορίζει την κατηγορία ανάλογα.

## Συνάρτηση **fix\_status**

Η συνάρτηση αυτή διορθώνει την τιμή της οικογενειακής κατάστασης (marital\_status). Επιστρέφει τιμές 'single' για ορισμένες τιμές του marital\_status, 'married' για την τιμή 'married', και 'unknown' για άλλες τιμές.

## Δημιουργία Πίνακα **CUSTOMERS**

Ο πίνακας **CUSTOMERS** δημιουργείται χρησιμοποιώντας τις προαναφερθείσες συναρτήσεις για τον υπολογισμό της ηλικίας, της οικογενειακής κατάστασης και του επιπέδου εισοδήματος για κάθε πελάτη από τον πίνακα **XSALES.CUSTOMERS**.

## Δημιουργία Πίνακα **PRODUCTS**

Ο πίνακας **PRODUCTS** περιέχει πληροφορίες προϊόντων από τον πίνακα **XSALES.PRODUCTS** με συμπληρωμένο το όνομα της κατηγορίας (CATEGORY\_NAME) χρησιμοποιώντας τον πίνακα **XSALES.CATEGORIES**.

## Δημιουργία Πίνακα **ORDERS**

Ο πίνακας **ORDERS** δημιουργείται από τους πίνακες **XSALES.ORDERS** και **XSALES.ORDER\_ITEMS**. Περιλαμβάνει τον αναγνωριστικό παραγγελίας, τον αναγνωριστικό πελάτη, το κανάλι, το προϊόν, την τιμή, το κόστος και τις ημέρες επεξεργασίας.

## 2ο Ερώτημα

### I) Συνάρτηση **CalculateMaxDelay**

1. Δημιουργούμε μια συνάρτηση `CalculateMaxDelay` που παίρνει ένα `order_id` και επιστρέφει το μέγιστο καθυστερημένο χρόνο για αυτήν την παραγγελία.
2. Χρησιμοποιούμε τον τελεστή `NVL` για να εξασφαλίσουμε ότι η επιστρεφόμενη τιμή είναι τουλάχιστον 0.

## II) Συνάρτηση `CalculateOrderProfit`

1. Δημιουργούμε μια συνάρτηση `CalculateOrderProfit` που παίρνει ένα `order_id` και επιστρέφει το τελικό κέρδος για αυτήν την παραγγελία.
2. Χρησιμοποιούμε τον τελεστή `NVL` για να εξασφαλίσουμε ότι η επιστρεφόμενη τιμή είναι τουλάχιστον 0.

## III) Η διαδικασία `ProcessOrders` καλείται με τη χρήση της εντολής `EXEC`.

### `ProcessOrders` Procedure:

- Δημιουργείται ένας κέρσοντας (`order_cursor`), ο οποίος επιλέγει πληροφορίες παραγγελίας από τον πίνακα `orders`.
- Ένας βρόγχος FOR χρησιμοποιείται για να περιηγηθεί σε κάθε παραγγελία και να υπολογίσει το κέρδος χρησιμοποιώντας τη συνάρτηση `CalculateOrderProfit`.
- Αν το κέρδος είναι αρνητικό, καταχωρείται στον πίνακα `deficit_orders`, αλλιώς στον πίνακα `profit_orders`.

### Ενδεικτικές Ερωτήσεις:

- Χρησιμοποιούνται υποδείξεις `INDEX` για τη βελτιστοποίηση των ερωτημάτων.
- Πραγματοποιείται επανευρετήριση των πινάκων `profit_orders` και `deficit_orders` μετά την εκτέλεση της διαδικασίας.

## IV.a: Συνολικά Έσοδα ανά Φύλο:

- Υπολογίζεται το συνολικό έσοδο ανά φύλο, αγνοώντας τυχόν διπλότυπες εγγραφές στον πίνακα `profit_orders`.

#### IV.b: Συνολικές Ζημιές ανά Φύλο:

- Υπολογίζονται οι συνολικές ζημιές ανά φύλο, αγνοώντας τυχόν διπλότυπες εγγραφές στον πίνακα `deficit_orders`.

#### V Συνολικά Έσοδα και Ζημιές ανά Κανάλι:

- Υπολογίζονται τα συνολικά έσοδα και ζημιές ανά κανάλι παραγγελιών, χρησιμοποιώντας τους πίνακες `profit_orders` και `deficit_orders`.

#### Παρατηρήσεις:

- Χρησιμοποιείτε τον τελεστή `/*+INDEX (IDX_ORDER)*/` στη δήλωση `SELECT` στη διαδικασία `ProcessOrders` για να χρησιμοποιήσετε τον δείκτη `IDX_ORDER`.

## 3ο Ερώτημα

1)

```
-- 3ο ΕΡΩΤΗΜΑ
DELETE FROM PLAN_TABLE;
EXPLAIN PLAN FOR
SELECT o.order_id, o.price - o.costs, o.days_to_process
FROM products p
JOIN orders o ON o.product_id = p.product_id
JOIN customers c ON o.customer_id = c.customer_id
WHERE p.category_name = 'Accessories'
AND o.channel = 'Internet'
AND c.gender = 'Male'
AND c.income_level = 'high'
AND o.days_to_process = 0;

-- Εκτύπωση του σχεδίου εκτέλεσης
SELECT OPERATION, OPTIONS, OBJECT_NAME, OBJECT_TYPE, ID, PARENT_ID, DEPTH,
COST, CPU_COST, IO_COST, CARDINALITY, FILTER_PREDICATES, ACCESS_PREDICATES, PROJECTION
FROM PLAN_TABLE
CONNECT BY PRIOR ID = PARENT_ID
START WITH ID=0
ORDER BY ID;
```

| ID | OPERATION        | OPTIONS | OBJECT_NAME | OBJECT_TYPE | ID | PARENT_ID | DEPTH | COST | CPU_COST  | IO_COST | CARDINALITY | FILTER_PREDICATES                                    | ACCESS_PREDICATES                   | P      |
|----|------------------|---------|-------------|-------------|----|-----------|-------|------|-----------|---------|-------------|--|-------------------------------------|--------|
| 1  | SELECT STATEMENT | (null)  | (null)      | (null)      | 0  | (null)    | 0     | 1498 | 360358231 | 1489    | 1597        | (null)   | (null)                              | (null) |
| 2  | HASH JOIN        | (null)  | (null)      | (null)      | 1  | 0         | 1     | 1498 | 360358231 | 1489    | 1597        | (null)   | "O"."CUSTOMER_ID"="C"."CUSTOMER_ID" | (sk)   |
| 3  | HASH JOIN        | (null)  | (null)      | (null)      | 2  | 1         | 2     | 1417 | 336337639 | 1409    | 1597        | (null)   | "O"."PRODUCT_ID"="P"."PRODUCT_ID"   | (sk)   |
| 4  | TABLE ACCESS     | FULL    | PRODUCTS    | TABLE       | 3  | 2         | 3     | 3    | 45766     | 3       | 10          | "P"."CATEGORY_NAME"='Accessories'                    | (null)                              | (rc)   |
| 5  | TABLE ACCESS     | FULL    | ORDERS      | TABLE       | 4  | 2         | 3     | 1414 | 334540273 | 1406    | 11501       | "O"."DAYS_TO_PROCESS"=0 AND "O"."CHANNEL"='Internet' | (null)                              | (rc)   |
| 6  | TABLE ACCESS     | FULL    | CUSTOMERS   | TABLE       | 5  | 1         | 2     | 80   | 19463943  | 80      | 37171       | "C"."GENDER"='Male' AND "C"."INCOME_LEVEL"='high'    | (null)                              | (rc)   |

a)  $all\_cost = w1 * cpu\_cost + w2 * io\_cost$  όπου  $w1=0,0001$  &  $w2=1 \Rightarrow$

$all\_cost = 0,0001 * 360358231 + 1 * 1489 = 36035,8231 + 1489 = 37524.8231$

b)

| id | cost | cpu_cost  | io_cost |
|----|------|-----------|---------|
| 0  | 0    | 0         | 0       |
| 1  | 1    | 4556649   | 0       |
| 2  | 0    | 1751600   | 0       |
| 3  | 3    | 45766     | 3       |
| 4  | 1414 | 334540273 | 1406    |
| 5  | 80   | 19463943  | 80      |



c)Πιο χρονοβόρα ενέργεια είναι η τέταρτη ενεργεια,δηλαδή το TABLE ACCESS FULL ORDERS ,καθώς έχει τις περισσότερες πλειάδες προς επεξεργασία

2)

a)

Το εκτιμώμενο πλήθος αποτελεσμάτων για το ερώτημα είναι 1597 πλειάδες ,σύμφωνα με το cardinality - -(Η εκτίμηση του cardinality βασίζεται στα στατιστικά των πινάκων και των στηλών που συμμετέχουν στο ερώτημα. Το cardinality δείχνει πόσες γραμμές αναμένεται να επιστραφούν μετά την εκτέλεση του ερωτήματος. ) - - Εμεις θέλουμε το cardinality για το select statement operations γιατί είναι το operation που κάνει εμφάνιση πινάκων

b) Για να βρούμε το πλήθος των πραγματικών πλειάδων που επιστρέφονται χρειάζεται να εκτελέσουμε την εξής εντολή :

```
SELECT COUNT(*)
```

```
FROM(
```

```
SELECT o.order_id, o.price - o.costs, o.days_to_process
```

```
FROM products p
```

```
JOIN orders o ON o.product_id = p.product_id
```

```
JOIN customers c ON o.customer_id = c.customer_id
```

```
WHERE p.category_name = 'Accessories'
```

```
AND o.channel = 'Internet'
```

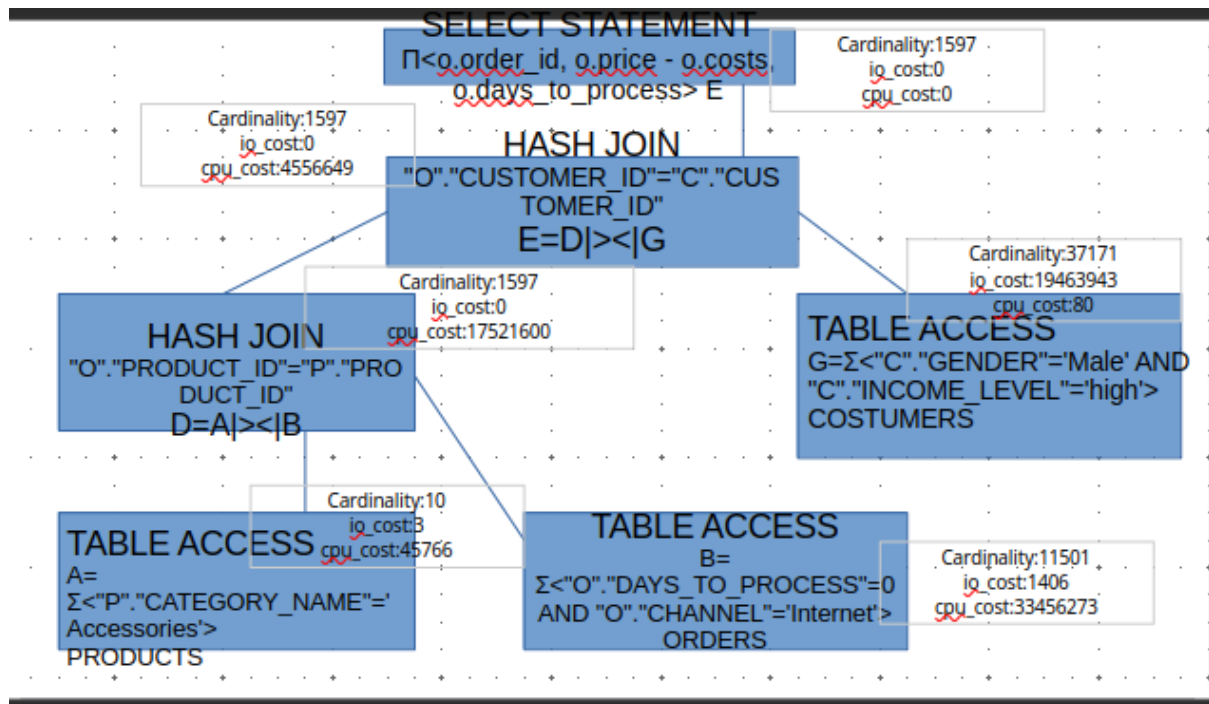
```
AND c.gender = 'Male'
```

```
AND c.income_level = 'high'
```

```
AND o.days_to_process = 0);
```

Όπου το αποτέλεσμα είναι 1891

c)



3)

Για την βελτιστοποίηση του ερωτήματος χρειάζεται να χρησιμοποιήσουμε btree ευρετήρια .Ευρετήρια που θα φτιάξουμε θα είναι για τα πεδία που περιέχονται στην συνθήκη where και στα join.

Ετσι δημιουργούμε τα παρακάτω ευρετήρια :

- CREATE INDEX idx\_orders\_product\_channel ON orders(product\_id, channel);
- CREATE INDEX idx\_products\_categoryname ON products(category\_name);
- CREATE INDEX idx\_customers\_id\_gender\_income ON customers(customer\_id, gender, income\_level);

και έχουμε την παρακάτω αλλαγή του optimizer.

| <pre> -- EXPLAIN PLAN FOR SELECT /*+INDEX(p idx_products_categoryname) INDEX(o idx_products_channel) INDEX(c idx_customers_id_gender_income)*/ o.order_id, o.price - o.costs, o.days_to_process FROM products p JOIN orders o ON o.product_id = p.product_id JOIN customers c ON o.customer_id = c.customer_id WHERE p.category_name = 'Accessories' AND o.channel = 'Internet' AND c.gender = 'Male' AND c.income_level = 'high' AND o.days_to_process = 0;  -- SELECT OPERATION, OPTIONS, OBJECT_NAME, OBJECT TYPE, ID, PARENT_ID, DEPTH, -- COST, CPU_COST, IO_COST, CARDINALITY, FILTER_PREDICATES, ACCESS_PREDICATES, PROJECTION -- FROM PLAN_TABLE -- CONNECT BY PRIOR ID = PARENT_ID -- START WITH ID=0 -- ORDER BY ID; </pre> |                        |              |             |    |           |       |        |           |         |  |                   |  |
|---|------------------------|--------------|-------------|----|-----------|-------|--------|-----------|---------|--|-------------------|--|
| OPERATION   | OPTIONS                | OBJECT_NAME  | OBJECT_TYPE | ID | PARENT_ID | DEPTH | COST   | CPU_COST  | IO_COST | CARDINALITY  | FILTER_PREDICATES |  |
| 1 SELECT STATEMENT  | (null)                 | (null)       | (null)      | 0  | (null)    | 0     | 1624   | 357884117 | 1615    | 1542 (null)  |                   |  |
| 2 HASH JOIN   | (null)                 | (null)       | (null)      | 1  | 0         | 1     | 1624   | 357884117 | 1615    | 1542 (null)  |                   |  |
| 3 NESTED LOOPS  | (null)                 | (null)       | (null)      | 2  | 1         | 2     | 1624   | 357884117 | 1615    | 1542 (null)  |                   |  |
| 4 STATISTICS COLLECTOR  | (null)                 | (null)       | (null)      | 3  | 2         | 3     | (null) | (null)    | (null)  | (null)   | (null)            |  |
| 5 HASH JOIN   | (null)                 | (null)       | (null)      | 4  | 3         | 4     | 1417   | 336134975 | 1409    | 1542 (null)  |                   |  |
| 6 NESTED LOOPS  | (null)                 | (null)       | (null)      | 5  | 4         | 5     | 1417   | 336134975 | 1409    | 1542 (null)  |                   |  |
| 7 STATISTICS COLLECTOR  | (null)                 | (null)       | (null)      | 6  | 5         | 6     | (null) | (null)    | (null)  | (null)   | (null)            |  |
| 8 TABLE ACCESS  | FULL                   | PRODUCTS     | TABLE       | 7  | 6         | 7     | 3      | 45766     | 3       | 10 "P"."CATEGORY_NAME"='Accessories'                 |                   |  |
| 9 TABLE ACCESS  | BY INDEX ROWID BATCHED | ORDERS       | TABLE       | 8  | 5         | 6     | 1414   | 334377609 | 1406    | 154 "O"."DAYS_TO_PROCESS"=0                          |                   |  |
| 10 INDEX  | RANGE SCAN             | IDX_ORDER... | INDEX       | 9  | 8         | 7     | (null) | (null)    | (null)  | (null)   | (null)            |  |
| 11 TABLE ACCESS   | FULL                   | ORDERS       | TABLE       | 10 | 4         | 5     | 1414   | 334377609 | 1406    | 11101 "O"."DAYS_TO_PROCESS"=0 AND "O"."CHANNEL"='I'  |                   |  |
| 12 INDEX  | RANGE SCAN             | IDX_CUSTO... | INDEX       | 11 | 2         | 3     | 206    | 17200742  | 206     | 1 (null)   |                   |  |
| 13 INDEX  | FULL SCAN              | IDX_CUSTO... | INDEX       | 12 | 1         | 2     | 206    | 17200742  | 206     | 37171 "C"."GENDER"='Male' AND "C"."INCOME_LEVEL"='I' |                   |  |

\*Παρατήρηση:Ο optimizer επιλέγει κάθε φορά ποιο index θα χρησιμοποιήσει έτσι δεν χρησιμοποιεί όλα τα ευρετήρια κάθε φορά που εκτελεί ένα σχήμα.

Τώρα το συνολικο κόστος είναι:

$all\_cost = w1 * cpu\_cost + w2 * io\_cost$  όπου  $w1=0,0001$  &  $w2=1 \Rightarrow$

$all\_cost = 0,0001 * 357884117 + 1 * 1615 = 35788,4117 + 1615 = 37403.4117$

## 4ο Ερώτημα

α)ι)

Script Output X

Query Result X

Query Result 3 X

SQL | All Rows Fetched: 6 in 0.023 seconds

| OPERATION          | OPTIONS | OBJECT_NAME | OBJECT_TYPE | ID | PARENT_ID | DEPTH | COST | CPU_COST  | IO_COST | CARDINALITY | FILTER_PREDICATES                                      | ACCESS_PREDICATES                   | PROJECTION  |
|--------------------|---------|-------------|-------------|----|-----------|-------|------|-----------|---------|-------------|--|-------------------------------------|---|
| 1 SELECT STATEMENT | (null)  | (null)      | (null)      | 0  | (null)    | 0     | 1496 | 366695705 | 1489    | 7271        | (null)   | (null)                              | (null)  |
| 2 HASH JOIN        | (null)  | (null)      | (null)      | 1  | 0         | 1     | 1489 | 366695705 | 1489    | 7271        | (null)   | "O"."CUSTOMER_ID"="C"."CUSTOMER_ID" | (keys=1) "O"."ORDER_ID"[NUMBER, 22], "O"."COSTS"[NUMBER, 22], "O"."DAYS_TO_PROCESS" |
| 3 HASH JOIN        | (null)  | (null)      | (null)      | 2  | 1         | 2     | 1418 | 341824012 | 1409    | 7271        | (null)   | "O"."PRODUCT_ID"="P"."PRODUCT_ID"   | (keys=1, rowset=200) "O"."ORDER_ID"[NUMBER, 22], "O"."COSTS"                        |
| 4 TABLE ACCESS     | FULL    | PRODUCTS    | TABLE       | 3  | 2         | 3     | 3    | 45766     | 3       | 10          | "P"."CATEGORY_NAME"="Accessories"                      | (null)                              | (rowset=32) "P"."PRODUCT_ID"[NUMBER, 22]  |
| 5 TABLE ACCESS     | FULL    | ORDERS      | TABLE       | 4  | 2         | 3     | 1414 | 335941446 | 1406    | 52353       | "O"."CHANNEL"="Internet" AND "O"."DAYS_TO_PROCESS">100 | (null)                              | (rowset=200) "O"."ORDER_ID"[NUMBER, 22], "O"."CUSTOMER_ID"                          |
| 6 TABLE ACCESS     | FULL    | CUSTOMERS   | TABLE       | 5  | 1         | 2     | 80   | 19463943  | 80      | 37171       | "C"."GENDER"="Male" AND "C"."INCOME_LEVEL"="high"      | (null)                              | (rowset=16) "C"."CUSTOMER_ID"[NUMBER, 22]   |

| id | cost | cpu_cost  | io_cost |
|----|------|-----------|---------|
| 0  | 0    | 0         | 0       |
| 1  | 1    | 5407750   | 0       |
| 2  | 0    | 5836800   | 0       |
| 3  | 3    | 45766     | 3       |
| 4  | 1414 | 335941446 | 1406    |
| 5  | 80   | 19463943  | 80      |

ii)

$\text{all\_cost} = w1 \cdot \text{cpu\_cost} + w2 \cdot \text{io\_cost}$  όπου  $w1=0,0001$  &  $w2=1 \Rightarrow$

$\text{all\_cost} = 0,0001 \cdot 366695705 + 1 \cdot 1489 = 38158.5705$

iii)

Πιο χρονοβόρα ενέργεια είναι η τέταρτη ενεργεια, δηλαδή το TABLE ACCESS FULL ORDERS , καθώς έχει τις περισσότερες πλειάδες προς επεξεργασία

b)

Για την βελτιστοποίηση του ερωτήματος χρησιμοποιούμε ευρετήρια btree index και bitmap index.

Χρησιμοποιούμε B-tree ευρετήρια όταν έχουμε στήλες με υψηλή καρδιναλότητα, δηλαδή όταν ο αριθμός των διακριτικών τιμών είναι σημαντικός σε σύγκριση με τον συνολικό αριθμό των εγγραφών. Τα B-tree ευρετήρια είναι κατάλληλα για συνθήκες ισότητας.

Αντίθετα, χρησιμοποιούμε Bitmap ευρετήρια όταν έχουμε στήλες με χαμηλή καρδιναλότητα, δηλαδή όταν ο αριθμός των διακριτικών τιμών είναι χαμηλός σε σύγκριση με τον συνολικό αριθμό των εγγραφών. Τα Bitmap ευρετήρια είναι αποτελεσματικά για συνθήκες εύρους τιμών, όπως στα ερωτήματα με πολλαπλές επιλογές συνθηκών.

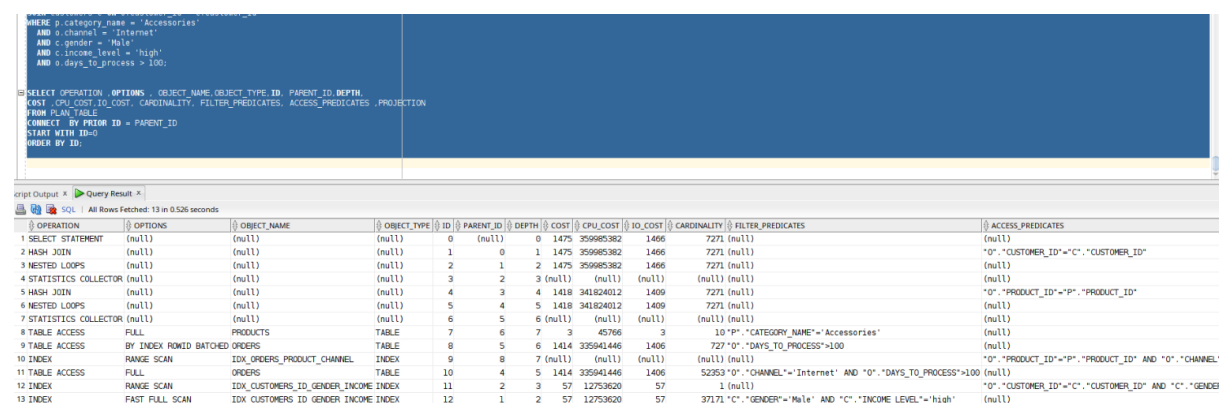
Για αυτό και δημιουργούμε τα αντίστοιχα btree και bitmap index:

```
CREATE BITMAP INDEX IDX_PRODUCTS_CATEGORY ON  
products(product_id, category_name);
```

```
CREATE BITMAP INDEX IDX_C_GEND ON customers(customer_id,  
gender);
```

```
CREATE BITMAP INDEX IDX_C_income ON customers(customer_id,  
income_level);
```

```
CREATE INDEX idx_orders_product_channel ON orders(order_id,product_id,  
channel);
```



```
WHERE p.category_name = 'Accessories'  
AND c.channel = 'Internet'  
AND c.gender = 'Male'  
AND c.income_level = 'high'  
AND o.days_to_process > 100;
```

SELECT OPERATION, OPTIONS, OBJECT\_NAME, OBJECT\_TYPE, ID, PARENT\_ID, DEPTH, COST, CPU\_COST, IO\_COST, CARDINALITY, FILTER\_PREDICATES, ACCESS\_PREDICATES  
FROM PLAN\_TABLE  
CONNECT BY PRIOR ID = PARENT\_ID  
START WITH ID=0  
ORDER BY ID;

| OPERATION              | OPTIONS                | OBJECT_NAME                          | OBJECT_TYPE | ID | PARENT_ID | DEPTH | COST   | CPU_COST  | IO_COST | CARDINALITY  | FILTER_PREDICATES                                    | ACCESS_PREDICATES |
|------------------------|------------------------|--------------------------------------|-------------|----|-----------|-------|--------|-----------|---------|--|--|-------------------|
| 1 SELECT STATEMENT     | (null)                 | (null)                               | (null)      | 0  | (null)    | 0     | 1475   | 359985382 | 1466    | 7271 (null)  | (null)   | (null)            |
| 2 HASH JOIN            | (null)                 | (null)                               | (null)      | 1  | 0         | 1     | 1475   | 359985382 | 1466    | 7271 (null)  | "O"."CUSTOMER_ID"="C"."CUSTOMER_ID"                  | (null)            |
| 3 NESTED LOOPS         | (null)                 | (null)                               | (null)      | 2  | 1         | 2     | 1475   | 359985382 | 1466    | 7271 (null)  | (null)   | (null)            |
| 4 STATISTICS COLLECTOR | (null)                 | (null)                               | (null)      | 3  | 2         | 3     | (null) | (null)    | (null)  | (null)   | (null)   | (null)            |
| 5 HASH JOIN            | (null)                 | (null)                               | (null)      | 4  | 3         | 4     | 1418   | 341824012 | 1409    | 7271 (null)  | "O"."PRODUCT_ID"="P"."PRODUCT_ID"                    | (null)            |
| 6 NESTED LOOPS         | (null)                 | (null)                               | (null)      | 5  | 4         | 5     | 1418   | 341824012 | 1409    | 7271 (null)  | (null)   | (null)            |
| 7 STATISTICS COLLECTOR | (null)                 | (null)                               | (null)      | 6  | 5         | 6     | (null) | (null)    | (null)  | (null)   | (null)   | (null)            |
| 8 TABLE ACCESS         | FULL                   | PRODUCTS                             | TABLE       | 7  | 6         | 7     | 3      | 45766     | 3       | 10 "P"."CATEGORY_NAME"='Accessories'                         | (null)   | (null)            |
| 9 TABLE ACCESS         | BY INDEX ROWID BATCHED | ORDERS                               | TABLE       | 8  | 5         | 6     | 1414   | 335941446 | 1406    | 727 "O"."DAYS_TO_PROCESS">100                                | (null)   | (null)            |
| 10 INDEX               | RANGE SCAN             | IDX_ORDERS_PRODUCT_CHANNEL           | INDEX       | 9  | 8         | 7     | (null) | (null)    | (null)  | (null)   | "O"."PRODUCT_ID"="P"."PRODUCT_ID" AND "O"."CHANNEL"  | (null)            |
| 11 TABLE ACCESS        | FULL                   | ORDERS                               | TABLE       | 10 | 4         | 5     | 1414   | 335941446 | 1406    | 52353 "O"."CHANNEL"='Internet' AND "O"."DAYS_TO_PROCESS">100 | (null)   | (null)            |
| 12 INDEX               | RANGE SCAN             | IDX_CUSTOMERS_ID_GENDER_INCOME INDEX | INDEX       | 11 | 2         | 3     | 57     | 12753620  | 57      | 1 (null)   | "O"."CUSTOMER_ID"="C"."CUSTOMER_ID" AND "C"."GENDER" | (null)            |
| 13 INDEX               | FAST FULL SCAN         | IDX_CUSTOMERS_ID_GENDER_INCOME INDEX | INDEX       | 12 | 1         | 2     | 57     | 12753620  | 57      | 37171 "C"."GENDER"='Male' AND "C"."INCOME_LEVEL"='high'      | (null)   | (null)            |

\*Παρατήρηση:Ο optimizer επιλέγει κάθε φορά ποιο index θα χρησιμοποιήσει έτσι δεν χρησιμοποιεί όλα τα ευρετήρια κάθε φορά που εκτελεί ένα σχήμα.

$\text{all\_cost} = w_1 \cdot \text{cpu\_cost} + w_2 \cdot \text{io\_cost}$  όπου  $w_1 = 0,0001$  &  $w_2 = 1 \Rightarrow$

$\text{all\_cost} = 0.0001 \times 359985382 + 1466 = 37464.5382$