1ο ερώτημα – Ευρετήριο με B+ δέντρο

1. Ξέρουμε ότι οι εγγραφές έχουν κατά μέσο όρο μέγεθος 200 bytes και ότι αποθηκεύονται σε block μεγέθους 1024 bytes με εκτεινόμενη καταχώρηση. Επομένως ο αριθμός των εγγραφών ανά block θα ισούται με:

$$bfr = \frac{1024}{200} = 5.12$$

Το bfr δεν στρογγυλοποιείται διότι έχουμε εκτεινόμενη καταχώρηση. Συνεπώς, το μέγεθος του αρχείου σε blocks θα ισούται με

$$l = \lceil \frac{\text{εγγραφές αρχείου}}{bfr} \rceil = \lceil \frac{15{,}000{,}000}{5.12} \rceil = 2,929,688 \text{ blocks.}$$

2. Από την θεωρία ξέρουμε ότι ένα κατακερματισμένο αρχείο πρέπει να έχει περίπου 25% έξτρα χώρο. Συνεπώς το μέγεθος αρχείου κατακερματισμού θα ισούται με:

$$l + l * 0.25 = l * 1.25 = 2,929,688 * 1.25 = 3,662,110 \text{ blocks}$$

3. Για να βρούμε το h αρκεί να διαιρέσουμε τους κόμβους κάθε επιπέδου με το αντίστοιχο bfr, μέχρι να φτάσουμε στην ρίζα. Αρχικά υπολογίζουμε τον αριθμό των εγγραφών ανά φύλλο, δηλαδή το bfr του τελευταίου επιπέδου:

Ξέρουμε ότι:
- το πεδίο κλειδιού αναζήτησης είναι 20 bytes
- κάθε δείκτης προς τις εγγραφές του αρχείου έχει μέγεθος 12 bytes
- κάθε φύλλο έχει ένα δείκτη προς επόμενο φύλλο μεγέθους 16 bytes
- όλα αυτά πρέπει να χωρέσουν σε ένα μπλοκ μεγέθους 1024 bytes.

Απ' τα παραπάνω προκύπτει η εξίσωση:

$$20bfr + 12bfr + 16 = 1024 \Leftrightarrow 32bfr = 1008 \Leftrightarrow bfr = \lfloor \frac{1008}{32} \rfloor \Leftrightarrow bfr = 31$$

Επειδή οι κόμβοι στο τελευταίο επίπεδο είναι πλήρεις, θα έχουμε $k = bfr * 100\% \Leftrightarrow k = 31$ εγγραφές ανά φύλλο.

Για το τελευταίο επίπεδο θέλουμε $\lceil \frac{\text{εγγραφές αρχείου}}{k} \rceil = \lceil \frac{15{,}000{,}000}{31} \rceil = 483,871$ κόμβους

Για να υπολογίσουμε τον αριθμό των εγγραφών ανά ενδιάμεσο κόμβο, αρκεί να λάβουμε υπόψη ότι τα ενδιάμεσα επίπεδα έχουν μόνο πεδία κλειδιών αναζήτησης (20 bytes) και δείκτες προς κόμβους ευρετηρίου (16 bytes). Άρα η εξίσωση θα είναι:

$$20bfr' + 16bfr' = 1024 \Leftrightarrow 36bfr' = 1024 \Leftrightarrow bfr' = \lfloor \frac{1024}{36} \rfloor \Leftrightarrow bfr' = 28$$

Επειδή το ευρετήριο έχει χτιστεί με χρήση B* δέντρου, οι κόμβοι θα είναι γεμάτοι κατά τα ⅔, επομένως έχουμε $k' = bfr' * 2/3 = 28 * 2/3 = 18$ εγγραφές ανά κόμβο.

Άρα το προτελευταίο επίπεδο έχει $\lceil \frac{blocks\ \text{επόμενου επιπέδου}}{k'} \rceil = \lceil \frac{483,871}{18} \rceil = 26,882$ κόμβους.

Για να βρούμε το h απλά επαναλαμβάνουμε την διαδικασία μέχρι να φτάσουμε στην ρίζα. Επομένως:

0ο επίπεδο: $\lceil \frac{5}{18} \rceil = 1$ κόμβος
1ο επίπεδο: $\lceil \frac{83}{18} \rceil = 5$ κόμβοι
2ο επίπεδο: $\lceil \frac{1494}{18} \rceil = 83$ κόμβοι
3ο επίπεδο: $\lceil \frac{26,882}{18} \rceil = 1494$ κόμβοι
4ο επίπεδο: $\lceil \frac{483,871}{18} \rceil = 26,882$ κόμβοι
5ο επίπεδο: $\lceil \frac{15,000,000}{31} \rceil = 483,871$ κόμβοι

Εφόσον η διαδικασία επαναλήφθηκε 6 φορές συνολικά, το ύψος του B* δέντρου (άρα και ο αριθμός των επιπέδων του) θα ισούται με $h = 6$.

4. Από το προηγούμενο ερώτημα έχουμε:

0ο επίπεδο: $\lceil \frac{5}{18} \rceil = 1$ κόμβος
1ο επίπεδο: $\lceil \frac{83}{18} \rceil = 5$ κόμβοι
2ο επίπεδο: $\lceil \frac{1494}{18} \rceil = 83$ κόμβοι
3ο επίπεδο: $\lceil \frac{26,882}{18} \rceil = 1494$ κόμβοι
4ο επίπεδο: $\lceil \frac{483,871}{18} \rceil = 26,882$ κόμβοι
5ο επίπεδο: $\lceil \frac{15,000,000}{31} \rceil = 483,871$ κόμβοι

Άρα, συνολικά το μέγεθος του ευρετηρίου είναι:

$s_i = 1 + 5 + 83 + 1494 + 26,882 + 483,871 = 512,336$ blocks

5. Αν το δέντρο γίνει B+ τότε το bfr θα μειωθεί. Πιο συγκεκριμένα, τα blocks θα είναι 50% γεμάτα αντί για 66% γεμάτα όπως στα B* δέντρα. Επειδή το τελευταίο επίπεδο εξακολουθεί να είναι 100% γεμάτο όπως υποδεικνύει η εκφώνηση, το μόνο που θα αλλάξει θα είναι το k' (αριθμός των εγγραφών/κόμβο των ενδιάμεσων επιπέδων).

Έχουμε λοιπόν $k' = bfr' * 1/2 = 28 * 1/2 = 14$

Ακολουθώντας την ίδια διαδικασία με προηγουμένως, βρίσκουμε πως:

0ο επίπεδο: $\lceil \frac{13}{14} \rceil = 1$ κόμβος

1ο επίπεδο: $\lceil \frac{177}{14} \rceil = 13$ κόμβοι

2ο επίπεδο: $\lceil \frac{2469}{14} \rceil = 177$ κόμβοι

3ο επίπεδο: $\lceil \frac{34,563}{14} \rceil = 2469$ κόμβοι

4ο επίπεδο: $\lceil \frac{483,871}{14} \rceil = 34,563$ κόμβοι

5ο επίπεδο: $\lceil \frac{15,000,000}{31} \rceil = 483,871$ κόμβοι

Άρα, συνολικά το μέγεθος του ευρετηρίου είναι:

$s_i' = 1 + 13 + 177 + 2469 + 34,563 + 483,871 = 521,094$ blocks

6. Επειδή το ευρετήριο έχει χτιστεί με χρήση B* δέντρου, το επίπεδο φύλλων θα αποθηκεύει τιμές με διπλότυπα τη μία δίπλα στην άλλη. Άρα αρκεί να κάνουμε μόνο μια αναζήτηση κόστους h στο B* δέντρο για να βρούμε και τις 20 τιμές.

   Ο αριθμός των εγγραφών ανά block του αρχείου ισούται με

   $$bfr = \lfloor \frac{S_B}{S_A} \rfloor = \lfloor \frac{1024}{200} \rfloor = 5 \text{ εγγραφές/block}$$

   Μπορούμε να διακρίνουμε 2 ακραίες περιπτώσεις:

   - Οι 20 τιμές είναι αποθηκευμένες σε $\frac{20}{bfr} = \frac{20}{5} = 4$ blocks στο αρχείο, επομένως το κόστος αναζήτησης θα είναι: $h + 4 = 6 + 4 = 10$
   - Κάθε τιμή από τις 20 είναι αποθηκευμένες σε ξεχωριστό block στο αρχείο, επομένως χρειαζόμαστε 20 διαφορετικές προσπελάσεις και το κόστος θα είναι: $h + 20 = 6 + 20 = 26$

   Επομένως, το κόστος αναζήτησης μπορεί να είναι οποιοσδήποτε ακέραιος στο διάστημα $[10, 26]$.

2ο ερώτημα – Δημιουργία και γέμισμα σχήματος

```
ALTER SESSION SET NLS_DATE_FORMAT='DD/MM/YY HH:MI';

------------------  Creation of table Customers  ------------------

CREATE TABLE Customers AS
SELECT
    id AS customer_id,
    gender,
    get_age_group(birth_date, SYSDATE) AS age_group,
    fix_status(marital_status) AS marital_status,
    get_income_level(income_level) AS income_level
FROM XSALES.customers;

DESC Customers;
SELECT * FROM Customers;

-------------------  Creation of table Products  -------------------

CREATE TABLE Products AS
SELECT
    p.identifier AS product_id,
    p.name AS productname,
    c.name AS categoryname,
    TO_NUMBER(REPLACE(p.list_price, ',', '.'), '9999.99') AS
    list_price
FROM XSALES.products p JOIN XSALES.categories c ON
p.subcategory_reference = c.id;

DESC Products;
SELECT * FROM Products;

--------------------  Creation of table Orders  --------------------

CREATE TABLE Orders AS
SELECT
    o.id AS order_id,
    i.product_id,
    o.customer_id,
    TO_NUMBER(TRUNC(SYSDATE) - TRUNC(i.order_date)) -
    TO_NUMBER(TRUNC(SYSDATE) - TRUNC(o.order_finished)) AS
    days_to_process,
    i.amount AS price,
    i.cost,
    o.channel
```

```
FROM XSALES.orders o
JOIN XSALES.order_items i ON o.id = i.order_id;

DESC Orders;
SELECT * FROM Orders;

--------------------------  Functions  --------------------------

-- get_age_group function
CREATE OR REPLACE FUNCTION get_age_group(birth_date DATE,
current_date DATE)
RETURN VARCHAR2
IS
    age NUMBER;
    age_group VARCHAR2(50);
BEGIN
    age := FLOOR(MONTHS_BETWEEN(current_date, birth_date) / 12);

    IF age < 40 THEN
      age_group := 'under 40';
    ELSIF age >= 40 AND age < 50 THEN
      age_group := '40-50';
    ELSIF age >= 50 AND age < 60 THEN
      age_group := '50-60';
    ELSIF age >= 60 AND age <= 70 THEN
      age_group := '60-70';
    ELSE
      age_group := 'above 70';
    END IF;

    RETURN age_group;
END;
/
```

```
-- get_income_level function
CREATE OR REPLACE FUNCTION get_income_level(income_level VARCHAR2)
RETURN VARCHAR2
IS
    cleaned_income_level VARCHAR2(50);
    income_value NUMBER;
    income_level2 VARCHAR2(50);
BEGIN
    -- Remove non-numeric characters and spaces
    cleaned_income_level := TRIM(TRANSLATE(income_level,
    'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz: ,',' '));

    -- Check if the cleaned input contains a hyphen
    -- indicating a range
    IF INSTR(cleaned_income_level, '-') > 0 THEN
        -- Split the cleaned input into two values
        income_value := TO_NUMBER(SUBSTR(cleaned_income_level,
        INSTR(cleaned_income_level, '-') + 1));
    ELSE
        -- If it's not a range, return the cleaned value as is
        income_value := TO_NUMBER(cleaned_income_level);
    END IF;

    IF income_value <= 129999 THEN
        income_level2 := 'low';
    ELSIF income_value <= 249999 THEN
        income_level2 := 'medium';
    ELSIF income_value >= 250000 THEN
        income_level2 := 'high';
    ELSE
        income_level2 := 'unknown';
    END IF;

    RETURN income_level2;
END;
/
```

```
-- fix_status function
CREATE OR REPLACE FUNCTION fix_status(marital_status VARCHAR2)
RETURN VARCHAR2
IS
    fixed_status VARCHAR2(50);
BEGIN
    CASE marital_status
        WHEN 'Widowed' THEN
            fixed_status := 'single';
        WHEN 'Separ.' THEN
            fixed_status := 'single';
        WHEN 'divorced' THEN
            fixed_status := 'single';
        WHEN 'NeverM' THEN
            fixed_status := 'single';
        WHEN 'Single' THEN
            fixed_status := 'single';
        WHEN 'single' THEN
            fixed_status := 'single';
        WHEN 'Divorc.' THEN
            fixed_status := 'single';
        ELSE
            fixed_status := 'married';
    END CASE;

    IF marital_status IS NULL THEN
        fixed_status := 'unknown';
    END IF;

    RETURN fixed_status;
END;
/
```

Ερώτημα 2ο – Εντοπισμός ζημιογόνων παραγγελιών

```
-- 2.1
CREATE TABLE delayed_orders AS
SELECT order_id, MAX(calculate_delay(days_to_process)) AS delay
FROM Orders GROUP BY order_id ORDER BY order_id;

SELECT * FROM delayed_orders;

CREATE OR REPLACE FUNCTION calculate_delay(days_to_process NUMBER)
RETURN NUMBER
IS
      delay NUMBER;
BEGIN
    -- calculate the delay
    delay := GREATEST(days_to_process - 20, 0);
    RETURN delay;
END;
/

-- 2.2
CREATE TABLE end_profits AS
SELECT e.order_id, SUM(o.price - o.cost - (p.list_price * 0.001 *
e.delay)) AS profit FROM
(
SELECT
    order_id,
    CASE
         WHEN MAX(days_to_process) - 20 < 0 THEN 0
         ELSE MAX(days_to_process) - 20
    END as delay
FROM Orders
GROUP BY order_id
) e JOIN Orders o ON e.order_id = o.order_id JOIN Products p ON
o.product_id = p.product_id GROUP BY e.order_id ORDER BY e.order_id;

SELECT * FROM end_profits;
```

```
-- 2.3
CREATE TABLE deficit(
    orderid NUMBER,
    customerid NUMBER,
    channel VARCHAR(20),
    deficit NUMBER
);

DESC deficit;
SELECT * FROM deficit;

CREATE TABLE profit(
    orderid NUMBER,
    customerid NUMBER,
    channel VARCHAR(20),
    profit NUMBER
);

DESC profit;
SELECT * FROM profit;

DECLARE
    -- Declare a cursor
    CURSOR c_cursor IS
    SELECT o.order_id, o.customer_id, o.channel,
    SUM(o.price - o.cost - (p.list_price * 0.001 * e.delay)) FROM
    (
    SELECT
        order_id,
        CASE
             WHEN MAX(days_to_process) - 20 < 0 THEN 0
             ELSE MAX(days_to_process) - 20
        END AS delay
    FROM Orders
    GROUP BY order_id
    ) e JOIN Orders o ON e.order_id = o.order_id
    JOIN Products p ON o.product_id = p.product_id
    GROUP BY o.order_id, o.customer_id, o.channel;

    -- Declare temporary variables
    temp_order_id Orders.order_id%TYPE;
    temp_customer_id Orders.customer_id%TYPE;
    temp_channel Orders.channel%TYPE;
    temp_end_profit NUMBER;
BEGIN
    OPEN c_cursor;
```

```
    LOOP
    FETCH c_cursor INTO temp_order_id, temp_customer_id,
    temp_channel, temp_end_profit;

    -- Exit the loop if no more rows to fetch
    EXIT WHEN c_cursor%NOTFOUND;

    IF temp_end_profit < 0 THEN
        -- If negative, insert into deficit table
        INSERT INTO deficit (orderid, customerid, channel, deficit)
        VALUES (temp_order_id, temp_customer_id, temp_channel,
        ABS(temp_end_profit));
    ELSIF temp_end_profit > 0 THEN
        -- If positive, insert into profit table
        INSERT INTO profit (orderid, customerid, channel, profit)
        VALUES (temp_order_id, temp_customer_id, temp_channel,
        temp_end_profit);
    END IF;

    END LOOP;
    CLOSE c_cursor;
END;
/

-- 2.4
SELECT pr.gender, pr.total_profit, df.total_deficit FROM
(SELECT c.gender, SUM(profit) AS total_profit FROM profit
JOIN Customers c ON customerid = c.customer_id GROUP BY c.gender) pr
JOIN
(SELECT c.gender, SUM(deficit) AS total_deficit FROM deficit
JOIN Customers c ON customerid = c.customer_id GROUP BY c.gender) df
ON pr.gender = df.gender;

-- 2.5
SELECT pr.channel, pr.total_profit, df.total_deficit FROM
(SELECT channel, SUM(profit) AS total_profit FROM profit GROUP BY
channel) pr
JOIN
(SELECT channel, SUM(deficit) AS total_deficit FROM deficit GROUP BY
channel) df
ON pr.channel = df.channel;
```

3ο Ερώτημα – Βελτιστοποίηση ερωτήματος ισότητας

Το πλάνο που επέλεξε ο optimizer έχει ως εξής:

1. Αρχικά κάνει ένα full scan των πινάκων orders και products για να βρει τις πλειάδες με o.days_to_process = 0, o.channel = 'Internet' και p.category_name = 'Accessories'.
2. Έπειτα κάνει τα αποτελέσματα hash join και διαβάζει τον πίνακα customers για να βρει τις πλειάδες με c.income_level = 'high' και c.gender = 'Male'.
3. Στην συνέχεια κάνει hash join τις πλειάδες που επέλεξε από τον πίνακα customers με το αποτέλεσμα του προηγούμενου hash join.
4. Τέλος κάνει select τα γνωρίσματα order_id, price-cost και days_to_process.

Ερωτήματα:

1. Σύμφωνα με το plan table:
   - IO_COST = 1488
   - CPU_COST = 350328701
   - Εκτιμώμενο συνολικό κόστος = 1497
   - Πιο χρονοβόρα ενέργεια = full table scan του πίνακα orders, με κόστος 1414 (επειδή ο πίνακας orders έχει πολύ περισσότερες πλειάδες από τους υπόλοιπους, και άρα παίρνει περισσότερο χρόνο για να διαβαστεί)

   Εντολές που χρησιμοποιήθηκαν:

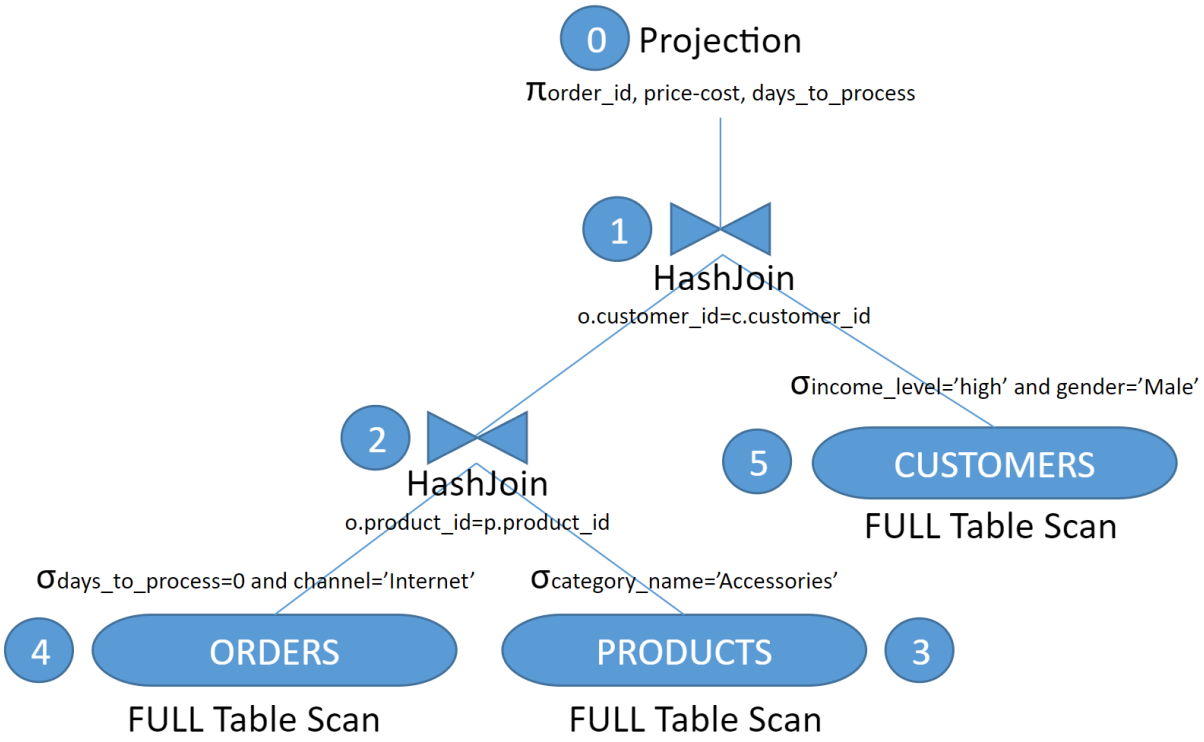
```
explain plan for
select order_id, price-cost,days_to_process from products p
join orders o on o.product_id=p.product_id join customers c on
o.customer_id=c.customer_id where p.categoryname='Accessories'
and o.channel='Internet' and c.gender='Male' and
c.income_level='high' and days_to_process=0;

SELECT id, parent_id, depth, operation, options, object_name,
access_predicates, filter_predicates, projection, cost,
cpu_cost, io_cost, cardinality FROM plan_table CONNECT BY
PRIOR id = parent_id START WITH id = 0 ORDER BY id;
```

COST	CPU_COST	IO_COST	CARDINALITY
1497	350328701	1488	10
1497	350328701	1488	10
1417	330755798	1409	10
3	45766	3	3
1414	330088582	1406	210
79	18277603	79	6938

- εκτιμώμενο συνολικό κόστος
- cpu cost
- io cost
- εκτιμώμενο πλήθος αποτελεσμάτων
- πιο χρονοβόρα ενέργεια

2. Σύμφωνα με το plan table:
   - Εκτιμώμενο πλήθος αποτελεσμάτων = 10
   - Αριθμός πραγματικών πλειάδων = 92
   - Πλάνο σε σχεσιακή άλγεβρα:
     1. $R_1 \leftarrow \sigma_{days\_to\_process=0\ and\ channel='Internet'}(orders)$
     2. $R_2 \leftarrow \sigma_{category\_name='Accessories'}(products)$
     3. $R_3 \leftarrow R_1 \bowtie_{o.product\_id=p.product\_id} R_2$
     4. $R_4 \leftarrow \sigma_{income\_level='high'\ and\ gender='Male'}(customers)$
     5. $R_5 \leftarrow R_3 \bowtie_{o.customer\_id=c.customer\_id} R_4$
     6. $\Pi_{R_5.order\_id,\ R_5.price-R_5.cost,\ R_5.days\_to\_process}(R_5)$



Εντολές που χρησιμοποιήθηκαν:

```
select count(*) as real_rows from
(select order_id, price-cost,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.categoryname='Accessories' and o.channel='Internet'
and c.gender='Male' and c.income_level='high' and
days_to_process=0);
```

	REAL_ROWS
1	92

3. Το τελικό κόστος μετά τη βελτιστοποίηση της σχεδίασης ισούται με 26 (δημιουργήθηκαν 3 ευρετήρια):

```
create index orders_idx on orders(days_to_process, channel,
customer_id, product_id);
create index customers_idx on customers(income_level, gender,
customer_id);
create index products_idx on products(categoryname,
product_id);
```

COST	CPU_COST	IO_COST	CARDINALITY
26	358937	26	10
26	358937	26	10
26	358937	26	10
(...	(null)	(n...	(null)
16	277223	16	10
16	277223	16	10
(...	(null)	(n...	(null)
1	7921	1	3
5	89767	5	3
2	67593	2	3
5	89767	5	3
2	67593	2	3
1	8171	1	1
1	8171	1	1

Το ευρετήριο

```
create index orders_idx on orders(days_to_process, channel,
customer_id, product_id);
```

είχε την μεγαλύτερη επίδραση στο κόστος, ρίχνοντάς το από 1497 σε 98.

COST	CPU_COST	IO_COST	CARDINALITY
98	19887971	97	10
98	19887971	97	10
18	315067	18	10
18	315067	18	10
3	45766	3	3
2	67593	2	3
5	89767	5	3
79	18277603	79	6938

Αν εξαιρέσουμε τα σύνθετα ευρετήρια, το πιο σημαντικό από όλα θα ήταν το

```
create index orders_idx on orders(days_to_process);
```

που κατάφερε να μειώσει το κόστος κατά 1030 μονάδες (από 1497 σε 467).

COST	CPU_COST	IO_COST	CARDINALITY
467	23301818	466	10
467	23301818	466	10
387	3728915	387	10
387	3728915	387	10
(...	(null)	(n...	(null)
(...	(null)	(n...	(null)
3	45766	3	3
4	154286	4	629
384	3061699	384	3
384	3061699	384	210
4	154286	4	629
79	18277603	79	6938

4ο Ερώτημα – Βελτιστοποίηση ερωτήματος ανισότητας

Ερωτήματα:

1. Σύμφωνα με το plan table:
   - IO_COST = 1488
   - CPU_COST = 395689204
   - Εκτιμώμενο συνολικό κόστος = 1498
   - Πιο χρονοβόρα ενέργεια = full table scan του πίνακα orders, με κόστος 1415 (επειδή ο πίνακας orders είναι πολύ μεγαλύτερος από τους υπόλοιπους)

   Εντολές που χρησιμοποιήθηκαν:

```
explain plan for
select order_id, price-cost,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.categoryname='Accessories' and o.channel='Internet'
and c.gender='Male' and c.income_level='high' and
days_to_process>100;

SELECT id, parent_id, depth, operation, options, object_name,
access_predicates, filter_predicates, projection, cost,
cpu_cost, io_cost, cardinality FROM plan_table CONNECT BY
PRIOR id = parent_id START WITH id = 0 ORDER BY id;
```

COST	CPU_COST	IO_COST	CARDINALITY
1498	395689204	1488	13541
1498	395689204	1488	13541
79	18277603	79	6938
1418	374416700	1409	13542
3	45766	3	3
1415	345331585	1406	284389

- εκτιμώμενο συνολικό κόστος
- cpu cost
- io cost
- πιο χρονοβόρα ενέργεια

2. Το τελικό κόστος μετά την βελτιστοποίηση της σχεδίασης ισούται με 1417 (δημιουργήθηκαν 2 unique ευρετήρια):

```
create unique index customers_idx on customers(income_level,
gender, customer_id);
create unique index products_idx on products(categoryname,
product_id);
```

COST	CPU_COST	IO_COST	CARDINALITY
1417	400101535	1407	13541
1417	400101535	1407	13541
1416	374371735	1407	13542
1	7921	1	3
1415	345324463	1406	284389
0	1900	0	1

Αν τα ευρετήρια δεν ήταν unique, η διαφορά στο κόστος θα ήταν μικρότερη σε σχέση με πριν (1444 αντί για 1417).

```
create index customers_idx on customers(income_level, gender,
customer_id);
create index products_idx on products(categoryname,
product_id);
```

COST	CPU_COST	IO_COST	CARDINALITY
1444	378946513	1434	13541
1444	378946513	1434	13541
27	1579879	27	6938
1416	374371735	1407	13542
1	7921	1	3
1415	345324463	1406	284389

Όλα τα άλλα ευρετήρια που δοκιμάσαμε (bitmap και μη) είχαν ελάχιστη ή καθόλου διαφορά στο κόστος.