Ευρετήρια και Κατακερματισμός

Β’ μέρος

ΣΥΝΟΨΗ ΕΝΟΤΗΤΑΣ

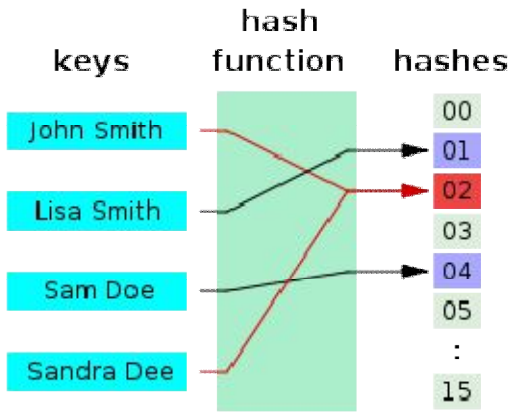
- Συναρτήσεις κατακερματισμού
- Κατακερματισμός στις βάσεις δεδομένων
- Στατικός vs. Δυναμικός Κατακερματισμός
- Bitmaps

Μειονεκτήματα Σειριακής Οργάνωσης Αρχείου

- Σειριακή οργάνωση αρχείου: Χρειαζόμαστε ευρετήριο ή δυαδική αναζήτηση για να εντοπίζουμε δεδομένα
  - Πολύ I/O
- Οργάνωση αρχείου με κατακερματισμό: Δεν απαιτεί ευρετήριο

Συνάρτηση Κατακερματισμού

- Συνάρτηση που αντιστοιχίζει δεδομένα μεταβλητού μήκους σε δεδομένα σταθερού μήκους
- Π.χ. ονόματα σε αριθμούς από το 1 έως το 15
- Χρήση: Hash table
- Αναζήτηση σε O(1)



Χρήσεις του Κατακερματισμού

- Ο Κατακερματισμός χρησιμοποιείται για την **οργάνωση των αρχείου** (εναλλακτικά των αρχείων σωρού ή των ταξινομημένων αρχείων)
- Ο Κατακερματισμός χρησιμοποιείται και για τη δημιουργία Δομών Ευρετηρίων (σε γνωρίσματα που δεν είναι Κλειδιά)
  - Σε αυτή την περίπτωση ονομάζεται **hash index** και οργανώνει τις τιμές των γνωρισμάτων (με τους αντίστοιχους δείκτες) σε ένα αρχείο κατακερματισμού.

Στατικός Κατακερματισμός

- Ένα **bucket** είναι μια μονάδα αποθήκευσης που περιέχει εγγραφές (συνήθως ένα block)
- Σε μια **hash οργάνωση αρχείου** λαμβάνουμε το bucket (δλδ τη διεύθυνση του block) στο οποίο ανήκει κάθε tuple απευθείας από το κλειδί αναζήτησης χρησιμοποιώντας μια συνάρτηση κατακερματισμού (**hash function**)

Συνάρτηση Κατακερματισμού στις Βάσεις

- Μια συνάρτηση κατακερματισμού *h* είναι μια συνάρτηση από το σύνολο όλων των κλειδιών αναζήτησης *K* στο σύνολο όλων των διευθύνσεων των buckets *B*
- Χρησιμοποιούνται για τον εντοπισμό εγγραφών για προσπέλαση, εισαγωγή ή διαγραφή
- Εγγραφές με διαφορετικά κλειδιά αναζήτησης μπορεί να αντιστοιχιστούν στο ίδιο bucket -> Μέσα στο κάθε bucket θα η εγγραφή θα πρέπει να αναζητηθεί σειριακά

Χαρακτηριστικά των hash functions

- Η χειρότερη hash function απεικονίζει όλα τα κλειδιά αναζήτησης στο ίδιο bucket
  - Χρόνος προσπέλασης ανάλογος του αριθμού των κλειδιών αναζήτησης
- Η ιδανική hash function
  - **Ομοιόμορφη**, αντιστοιχεί σε κάθε bucket τον ίδιο αριθμό κλειδιών αναζήτησης
  - **Τυχαίας κατανομής**, ώστε κάθε bucket να έχει τον ίδιο αριθμό εγγραφών ανεξάρτητα από την πραγματική κατανομή των τιμών των κλειδιών
- Συνήθως η hash function εκτελεί υπολογισμούς με τη δυαδική αναπαράσταση των κλειδιών αναζήτησης

Παράδειγμα

Παράδειγμα hash οργάνωσης αρχείου του *instructor*, χρησιμοποιώντας το *dept_name* ως κλειδί

- 8 buckets
- Η δυαδική αναπαράσταση του *i*-οστού χαρακτήρα θεωρείται ο ακέραιος του *i*
- Η συνάρτηση κατακερματισμού επιστρέφει το άθροισμα της δυαδικής αναπαράστασης των χαρακτήρων modulo 8
  - E.g. h(Music) = 1 h(History) = 2
    h(Physics) = 3 h(Elec. Eng.) = 3

Παράδειγμα

bucket 0

bucket 1

15151	Mozart	Music	40000

bucket 2

32343	El Said	History	80000
58583	Califieri	History	60000

bucket 3

22222	Einstein	Physics	95000
33456	Gold	Physics	87000
98345	Kim	Elec. Eng.	80000

bucket 4

12121	Wu	Finance	90000
76543	Singh	Finance	80000

bucket 5

76766	Crick	Biology	72000

bucket 6

10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

bucket 7

Χειρισμός υπερχείλισης buckets

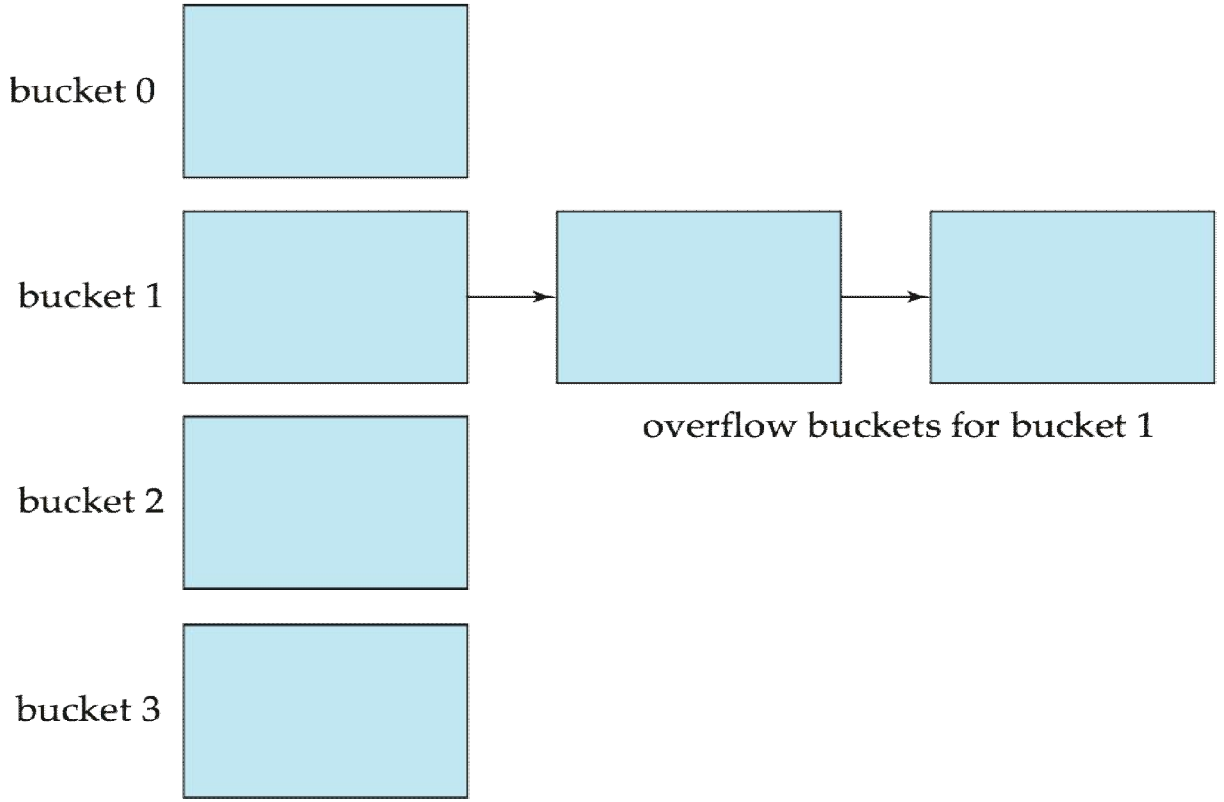
- Υπερχείλιση μπορεί να συμβεί:
  - Ανεπαρκής αριθμός buckets
  - Ασυμμετρία στην κατανομή των εγγραφών
    - » Πολλές εγγραφές έχουν το ίδιο κλειδί αναζήτησης
    - » Η επιλεγμένη hash function παράγει μη ομοιόμορφη κατανομή
- Η πιθανότητα υπερχείλισης μπορεί να μειωθεί αλλά όχι να εξαλειφθεί -> buckets υπερχείλισης (***overflow buckets***)

Εναλλακτικές

- Υπάρχουν τρεις βασικοί τρόποι αντιμετώπισης του Overflow:
  - ***Chaining (αλυσίδα υπερχείλισης):***
    αν το bucket **h**(v) είναι γεμάτο, βάλε σε αλυσίδα ένα άδειο bucket στο γεμάτο bucket για να το επεκτείνεις
  - ***Open Addressing (ανοικτή διευθυνσιοδότηση) :***
    αν το ***h***(v) είναι γεμάτο, βάλε την εγγραφή στο ***h***(v)+1 Αν και αυτό είναι γεμάτο, βάλε την στο ***h***(v)+2, κλπ.
  - ***Double-hashing (Διπλός / Πολλαπλός Κατακερματισμός):***
    Χρήση 2 συναρτήσεων (***h*** και ***h'***) αν το ***h***(v) είναι γεμάτο δοκίμασε το ***h'***(v). Αν το ***h'***(v) είναι επίσης γεμάτο, δοκίμασε κάποιο άλλο σχήμα (π.χ., μια τρίτη συνάρτηση κατακερματισμού, κλπ.)

Αλυσίδα υπερχείλισης

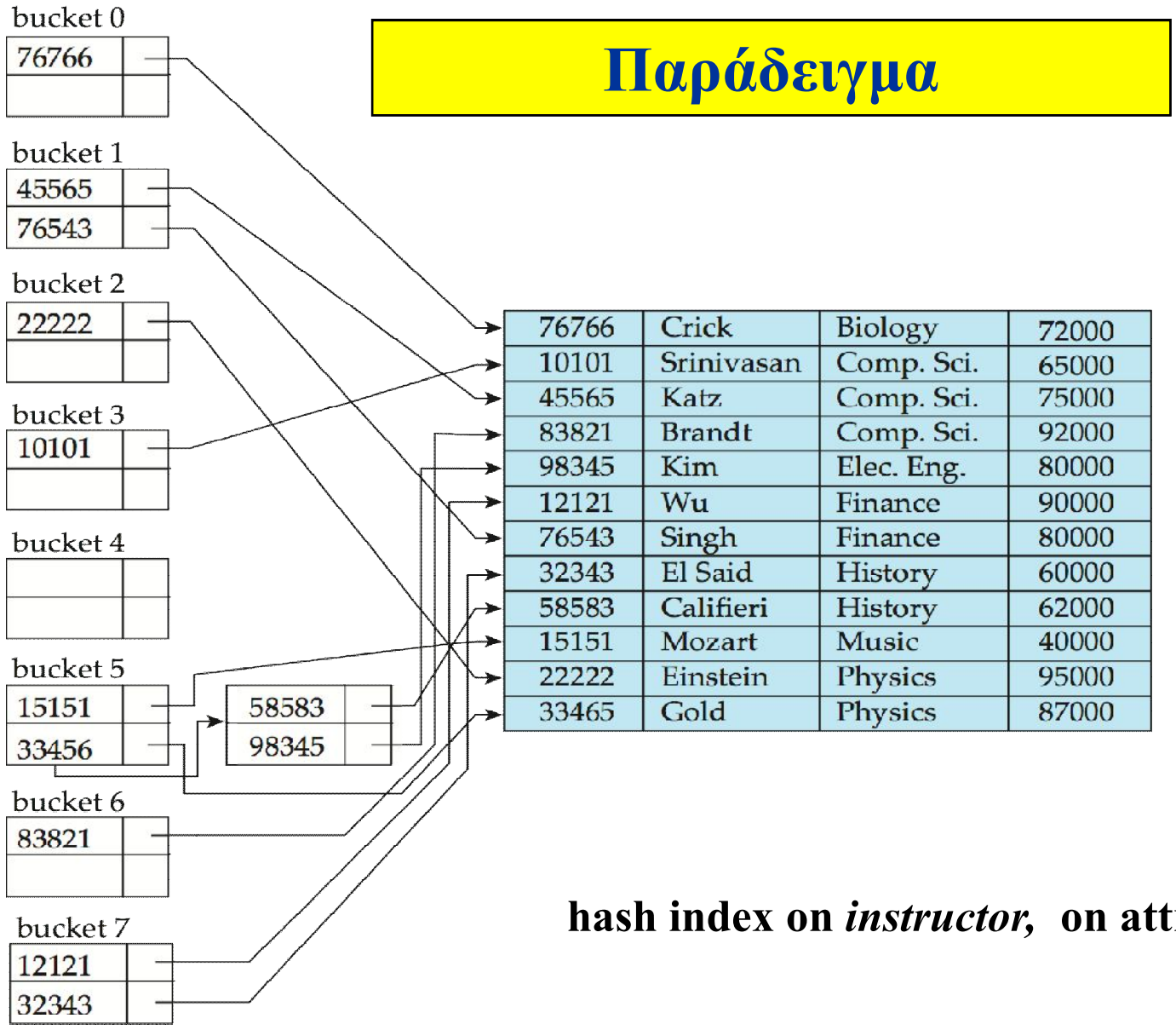
- **Αλυσίδα υπερχείλισης** – τα buckets υπερχείλισης συνδέονται σε συνδεδεμένη λίστα
- Ονομάζεται και κλειστό hashing



Ευρετήρια Hash

- Ένα ευρετήριο **hash** οργανώνει τα κλειδιά αναζήτησης με τους σχετιζόμενους δείκτες προς τις εγγραφές
- Είναι πάντα δευτερεύοντα ευρετήρια
  - Αν το ίδιο το αρχείο είναι οργανωμένο με hashing δε χρειάζεται ξεχωριστό πρωτεύον ευρετήριο για το ίδιο κλειδί αναζήτησης
  - Χρησιμοποιούμε τον όρο ευρετήριο hash και για δευτερεύον ευρετήριο και για αρχεία οργανωμένα με hash

Παράδειγμα



hash index on *instructor*, on attribute *ID*

Μειονεκτήματα στατικού hashing

- Στον στατικό κατακερματισμό η συνάρτηση hash h αντιστοιχίζει ένα κλειδί αναζήτησης σε ένα σταθερό σύνολο από B buckets. Οι βάσεις όμως αυξομειώνονται με τον χρόνο
  - Αν ο αρχικός αριθμός buckets είναι πολύ μικρός και η βάση μεγαλώνει, η απόδοση μειώνεται λόγω υπερχειλίσεων
  - Αν δεσμευτεί χώρος για πιθανή αύξηση της βάσης, χάνεται σημαντικός χώρος (ιδίως αρχικά, όταν τα buckets θα είναι άδεια)
  - Αν η βάση μειωθεί πάλι χάνεται χώρος
- Λύση: Περιοδική αναδιοργάνωση του αρχείου με νέα συνάρτηση hash
  - Χρονοβόρα διαδικασία, διακόπτει τη λειτουργία της βάσης
- Καλύτερη λύση: Επιτρέπουμε δυναμική αλλαγή του αριθμού των buckets.

Δυναμικό Hashing

- Ιδανικό για βάσεις
- Ο αριθμός των buckets αυξάνει/μειώνεται καθώς το αρχείο (σχέση) μεγαλώνει/μικραίνει
- Επεκτάσιμος κατακερματισμός – extendable hashing

Επεκτάσιμος Κατακερματισμός

- Η συνάρτηση ***h*** επιλέγεται ώστε το πεδίο τιμών της να είναι ένα ***πολύ μεγάλο*** σύνολο ακεραίων (π.χ., B = 2^b , όπου b=32)
- Ανά πάσα στιγμή χρησιμοποιείται μόνο ένα πρόθεμα του hash
  - Τα i πιο σημαντικά bits, $0 \leq i \leq 32$
- Ο i-bit αριθμός χρησιμοποιείται σαν ευρετήριο σε έναν πινακα που περιέχει δείκτες στα κατάλληλα buckets. Ο πίνακας λέγεται ότι έχει length i
- Ο πίνακας αποθηκεύεται στον δίσκο και αυξομειώνεται δυναμικά

Επεκτάσιμος Κατακερματισμός

- Πρόθεμα με i bits, $0 \le i \le 32$.
  - Ο πίνακας με διευθύνσεις bucket έχει μέγεθος = 2^{i}. Αρχικά $i = 0$
  - Η τιμή του i αυξάνεται και μειώνεται ακολουθώντας το μέγεθος της βάσης

- Οποιοσδήποτε αριθμός από (δύναμη του 2) γειτονικά στοιχεία στον πίνακα μπορεί να «δείχνει» στο ίδιο bucket. Οπότε ο πραγματικός αριθμός buckets είναι $< 2^{i}$

- Ο αριθμός των buckets αλλάζει δυναμικά λόγω της συνένωσης και της διαίρεσης των buckets.

Επεκτάσιμος Κατακερματισμός (2)

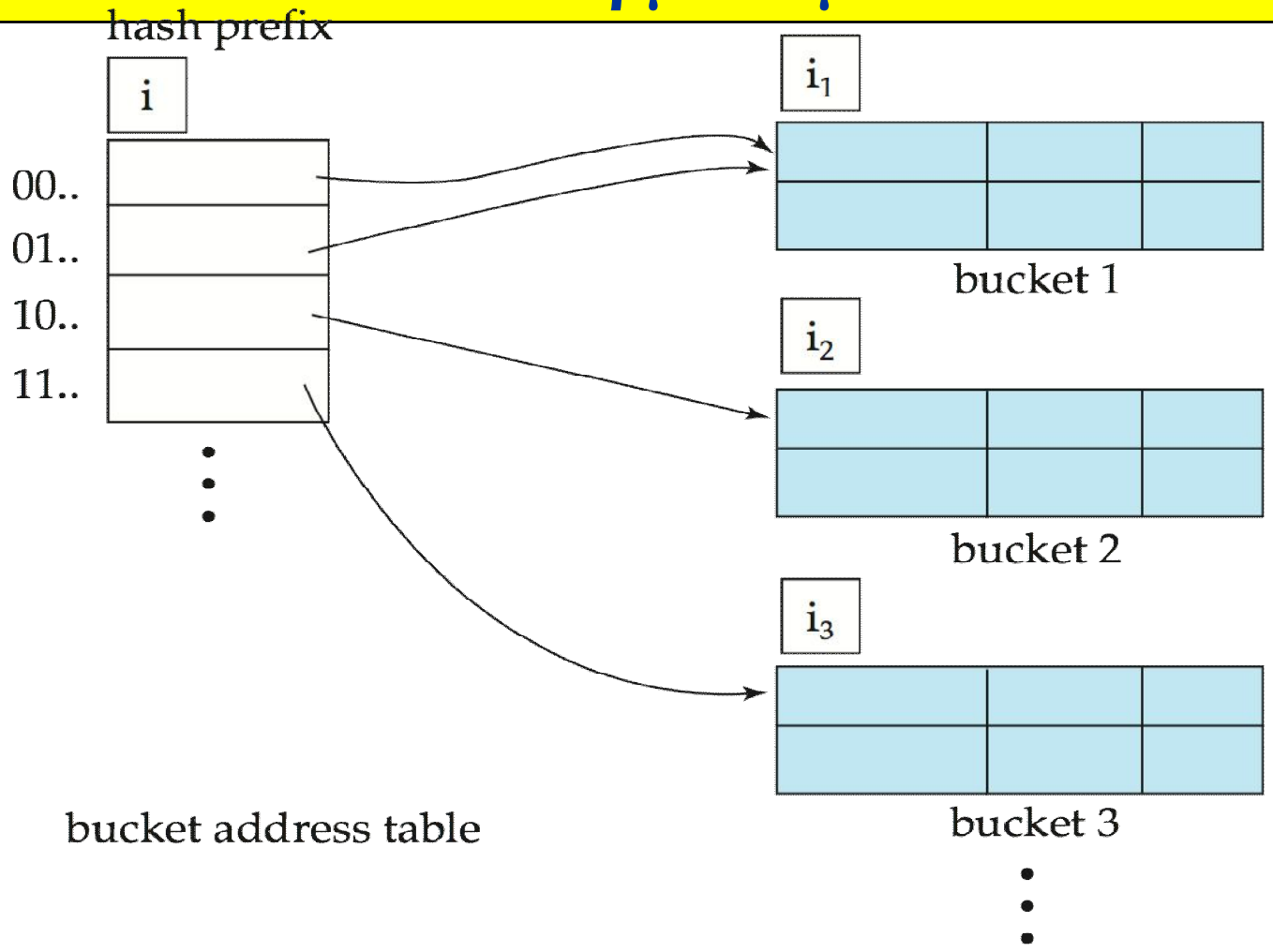
- Αν 2^k στοιχεία δείχνουν στο ίδιο bucket j, το μήκος του κοινού προθέματος του bucket ισούται με

$$i_j = i - k$$

- Οι τιμές των κλειδιών που κατακερματίζονται στο ***ίδιο*** bucket j έχουν τον ίδιο αριθμό i_j κοινών bits.

- ***Δεν απαιτείται*** περιοχή υπερχείλισης

Γενική δομή επεκτάσιμου κατακερματισμού



In this structure, $i_2 = i_3 = i$, whereas $i_1 = i - 1$

Χρήση της δομής

- Κάθε bucket j αποθηκεύει την τιμή i_j
  - Όλες οι εγγραφές του πίνακα που δείχνουν στο ίδιο bucket έχουν την ίδια τιμή για τα πρώτα i_j bits.
- Για την εύρεση του bucket που περιέχει το κλειδί K_j:
  1. Υπολογίζω το $h(K_j) = X$
  2. Χρησιμοποιώ τα πρώτα i bits του X για να βρω την θέση της εγγραφής στον πίνακα διευθύνσεων bucket και να ακολουθήσω τον δείκτη στο σωστό bucket
- Για εισαγωγή εγγραφής με κλειδί K_j
  - Βρίσκω το κατάλληλο bucket με αναζήτηση, έστω j
  - Αν υπάρχει χώρος στο bucket j εισάγω την εγγραφή
  - Αλλιώς διαιρώ το bucket και ξαναπροσπαθώ

Εισαγωγή στη δομή

Για διαίρεση του bucket *j* κατά την εισαγωγή τιμής K_j:

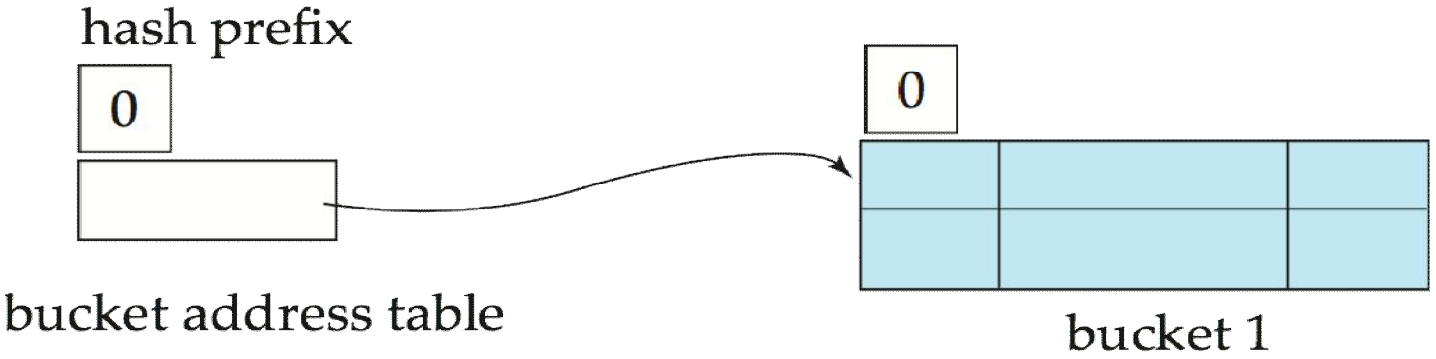
- Αν $i > i_j$ (υπάρχουν >1 δείκτες στο bucket j)
  - Δεσμεύω νεό bucket *z*, και θέτω $i_j = i_z = (i_j + 1)$
  - Ανανεώνω τις μισές εγγραφές του πίνακα διευθύνσεων bucket που έδειχναν στο *j* ώστε να δείχνουν στο *z*
  - Αφαιρώ όλες τις εγγραφές από το *j* και τις επανατοποθετώ (στο *j* ή στο *z*)
  - Υπολογίζω πάλι το bucket για το K_j και το εισάγω (διαιρώ ξανά αν χρειάζεται)
- Αν $i = i_j$ (μόνο ένας δείκτης στο bucket j)
  - Αν το *i* φτάσει κάποιο όριο *b*, ή αν έγιναν πολλές διαιρέσεις σε αυτήν την εισαγωγή τότε δημιούργησε ένα overflow bucket
  - Αλλιώς
    - » Αύξησε το *i* διπλασιάζοντας το μέγεθος του πίνακα διευθύνσεων
    - » Αντικατάστησε κάθε καταχώρηση του πίνακα με 2 που δείχνουν στο ίδιο bucket.
    - » Υπολόγισε ξανά την καταχώρηση στον πίνακα για το K_j
      Τώρα $i > i_j$, οπότε ακολούθησε τον πρώτο κανόνα

Παράδειγμα

dept_name	h(*dept_name*)
Biology	0010 1101 1111 1011 0010 1100 0011 0000
Comp. Sci.	1111 0001 0010 0100 1001 0011 0110 1101
Elec. Eng.	0100 0011 1010 1100 1100 0110 1101 1111
Finance	1010 0011 1010 0000 1100 0110 1001 1111
History	1100 0111 1110 1101 1011 1111 0011 1010
Music	0011 0101 1010 0110 1100 1001 1110 1011
Physics	1001 1000 0011 1111 1001 1100 0000 0001

Παράδειγμα (2)

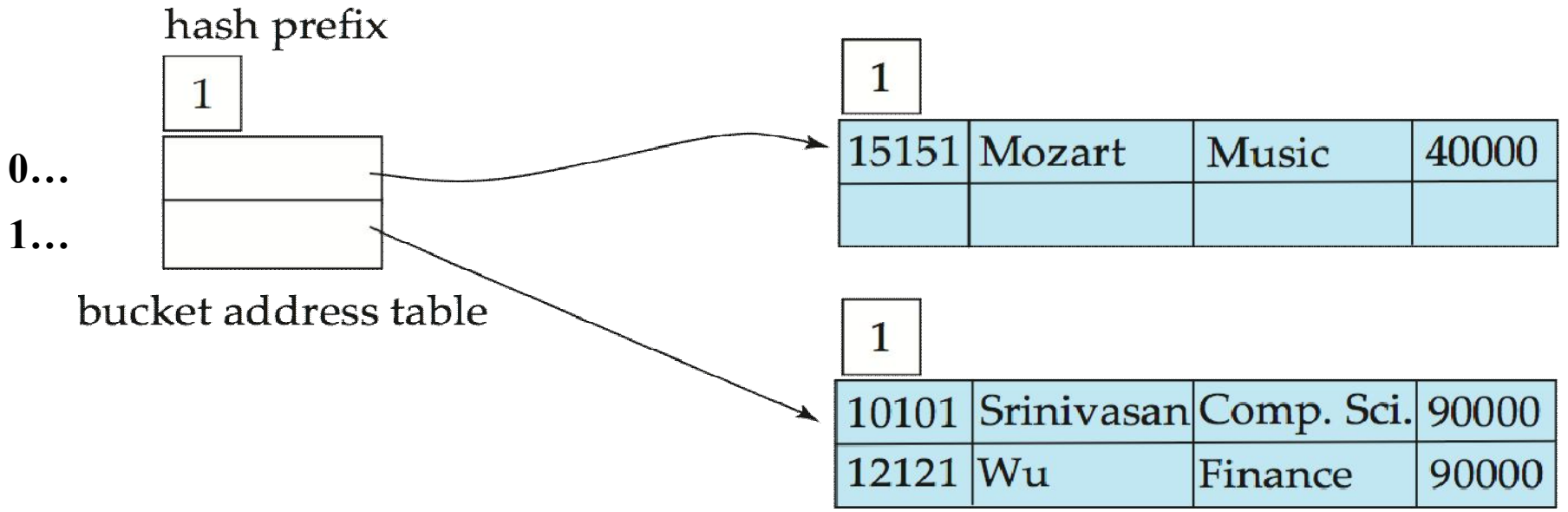
- **Initial Hash structure; bucket size = 2**



“Mozart” 0011…
“Srinivasan” 1111…
“Wu” 1010…

Παράδειγμα (3)

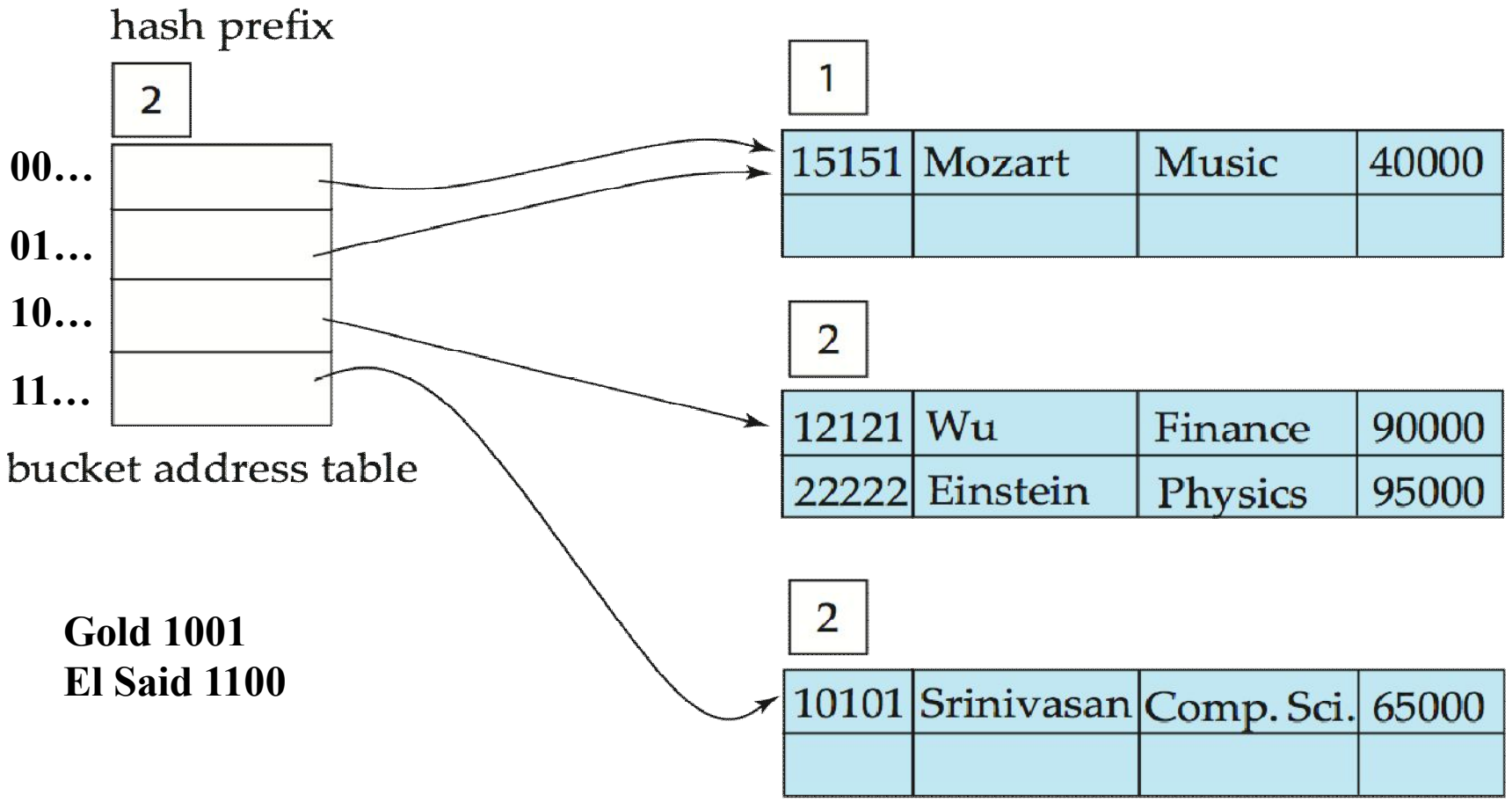
- **Hash structure after insertion of "Mozart", "Srinivasan", and "Wu" records**



Einstein 1001…

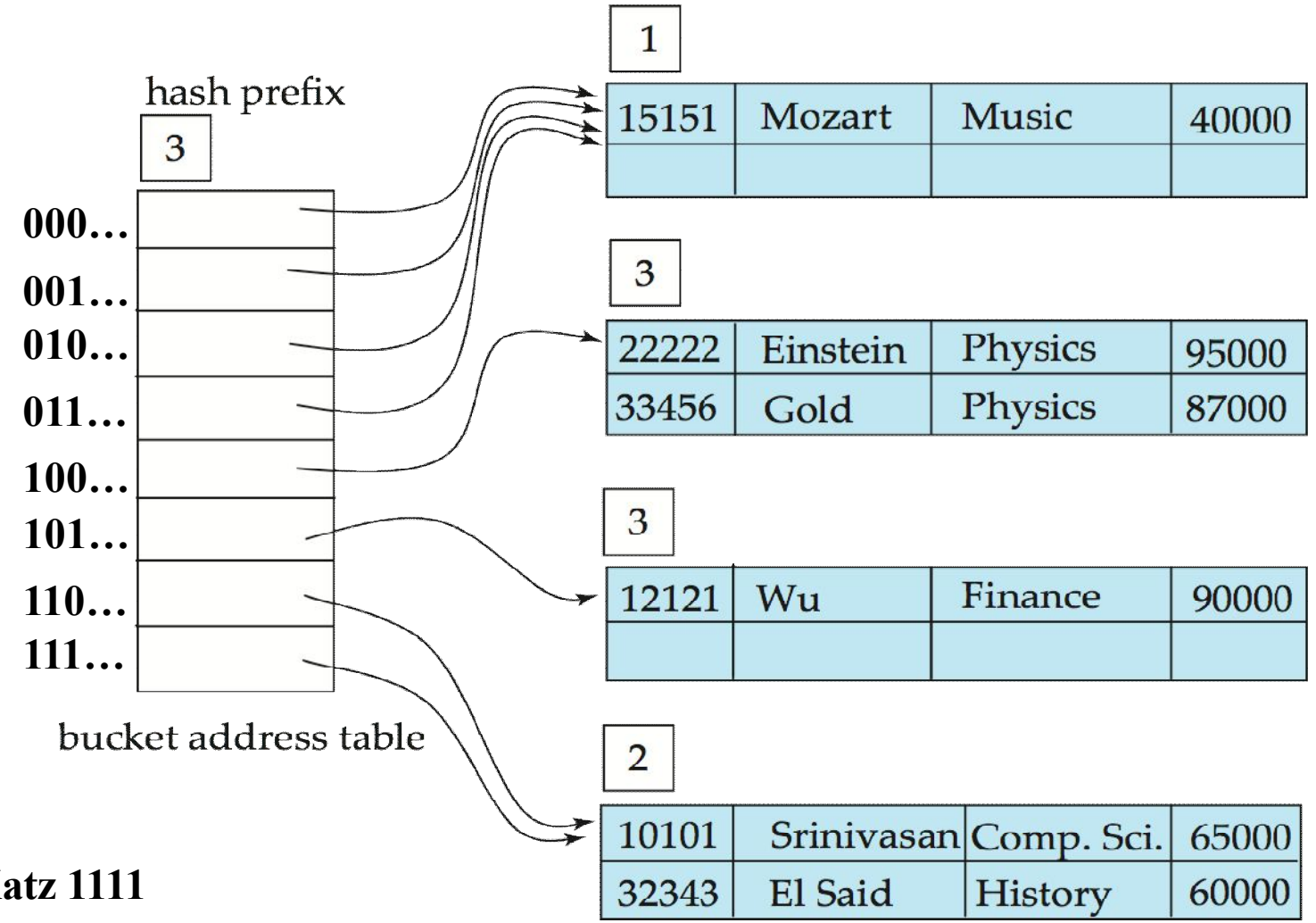
Παράδειγμα (4)

- **Hash structure after insertion of Einstein record**



Παράδειγμα (5)

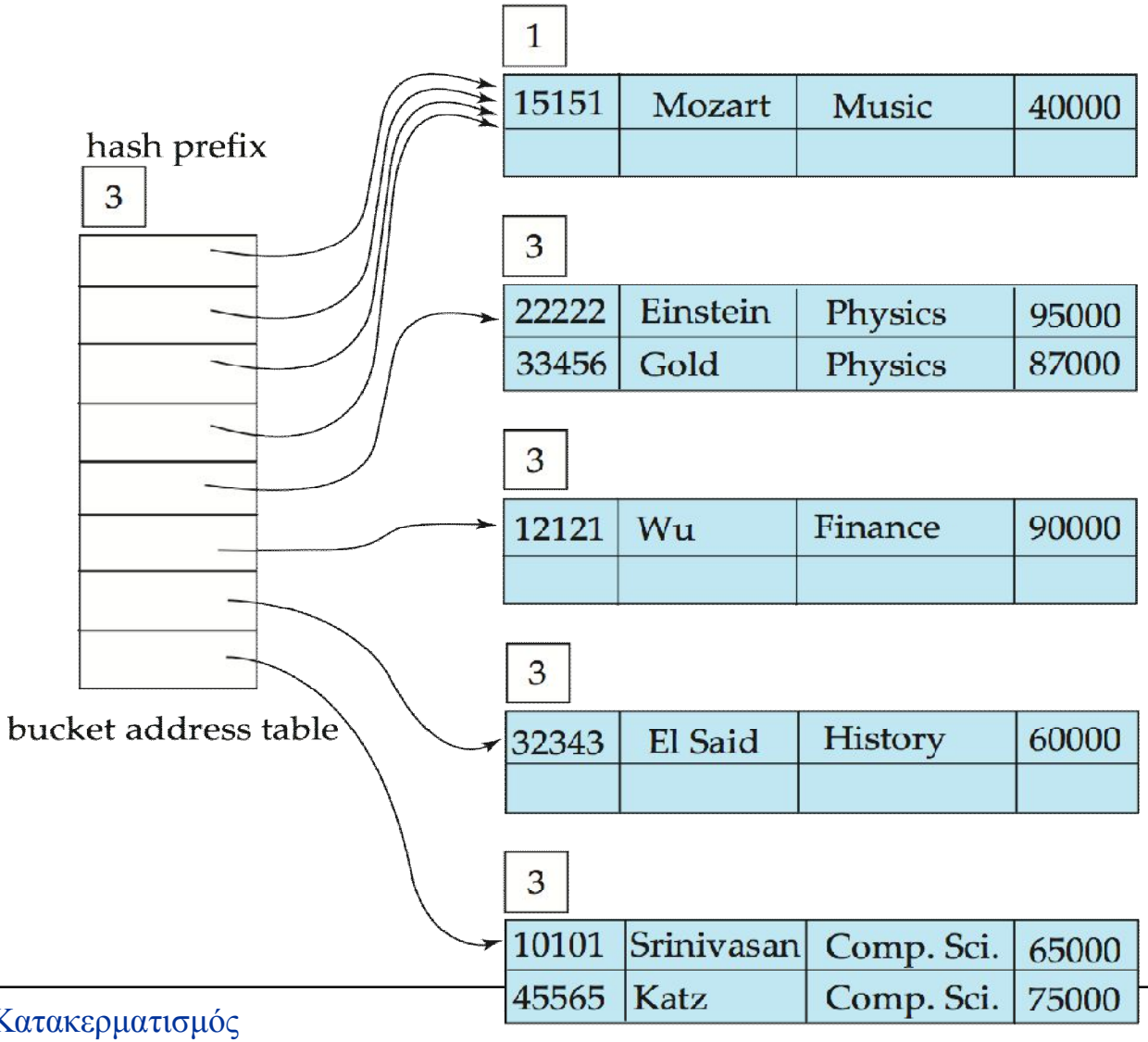
- **Hash structure after insertion of Gold and El Said records**



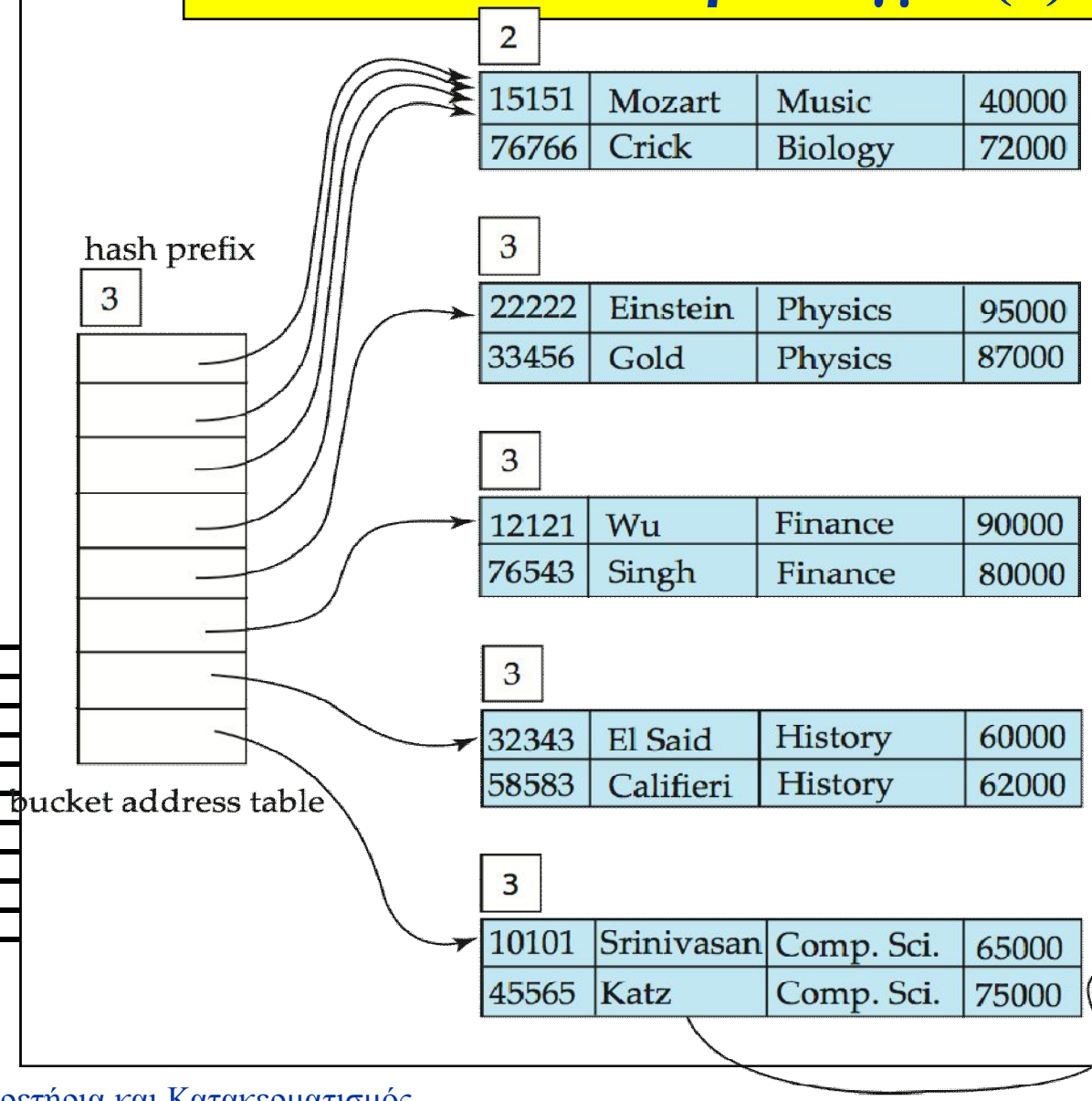
Katz 1111

Παράδειγμα (6)

- **Hash structure after insertion of Katz record**



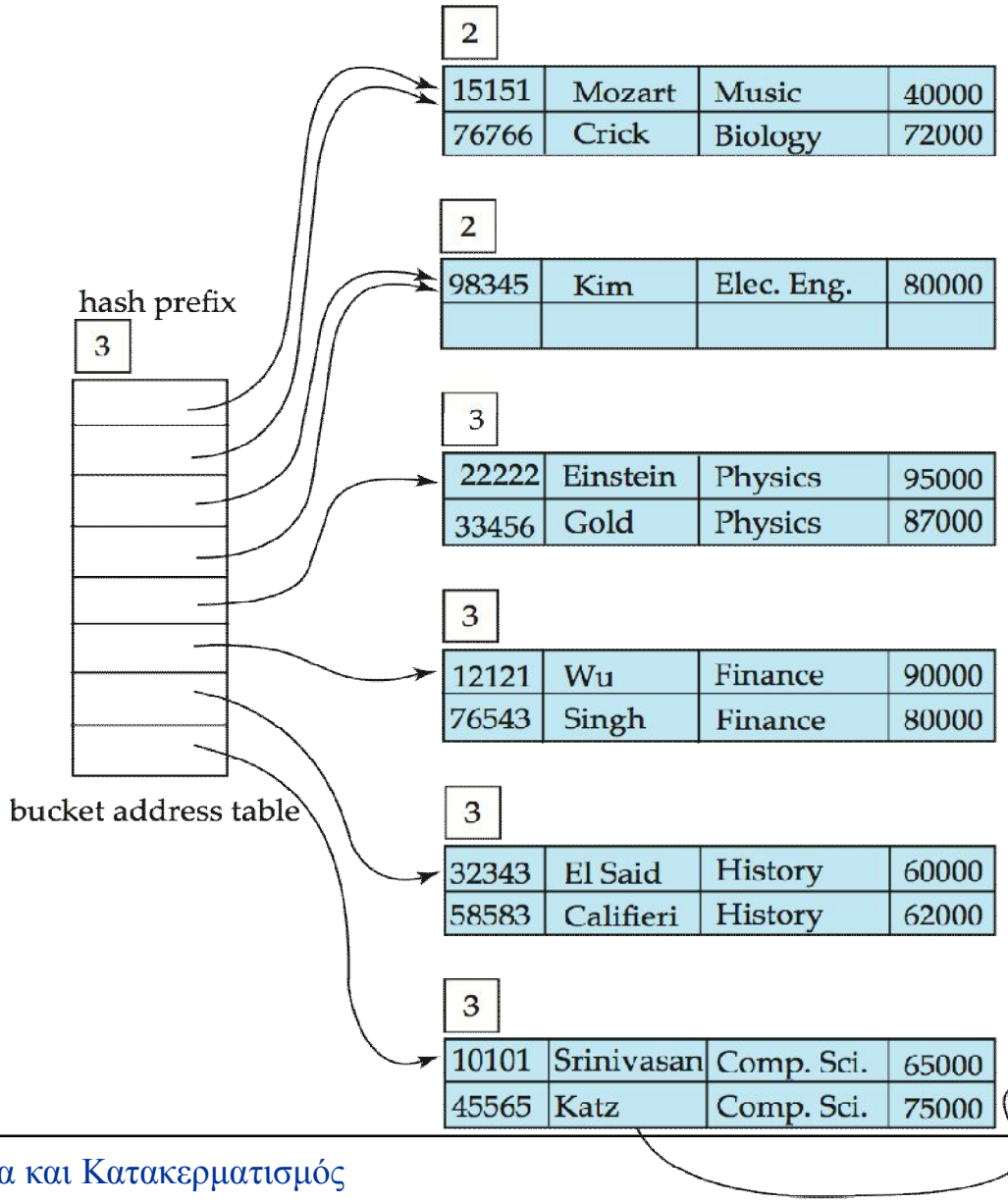
Παράδειγμα (7)



And after insertion of eleven records

Kim 0100…

Παράδειγμα (8)



And after insertion of Kim record in previous hash structure

Διαγραφή στη δομή

- Για διαγραφή κλειδιού
  - Βρες το στο bucket του και αφαιρεσέ το
  - Αφαίρεσε το bucket αν αδειάσει (με κατάλληλες ανανεώσεις στον πίνακα).
  - Μπορεί να γίνει συνένωση με άλλο bucket μόνο αν έχουν την ίδια τιμή i_j και το ίδιο πρόθεμα i_j –1
  - Μπορεί να συμβεί και μείωση του μεγέθους του πίνακα διευθύνσεων
    - » Πολύ ακριβή διαδικασία και πρέπει να γίνεται μόνο αν ο αριθμός των buckets γίνει πολύ μικρότερος του μεγέθους του πίνακα

Επεκτάσιμος Κατακερματισμός

- **ΠΛΕΟΝΕΚΤΗΜΑΤΑ**
  - Η απόδοση των ανακλήσεων είναι **σταθερή** καθώς το αρχείο μεγαλώνει
- **ΜΕΙΟΝΕΚΤΗΜΑΤΑ**
  - Οι ενημερώσεις είναι ακριβές, ειδικά όταν ο κατάλογος διπλασιάζεται
  - Ο κατάλογος απαιτεί μνήμη / χώρο
  - Αν ο κατάλογος μεγαλώσει ώστε να μη χωρά στη μνήμη, οι ανακλήσεις εγγραφών πλέον απαιτούν δύο I/O πράξεις
  - Αν ένας κάδος υπερχειλίσει **έχοντας τις ίδιες τιμές κλειδιού**, τότε ο επεκτάσιμος κατακερματισμός θα σπάει αυτό το bucket για πάντα

Ταξινομημένα Ευρετήρια vs. Hashing

- Κόστος περιοδικών αναδιοργανώσεων
- Συχνότητα εισαγωγών και διαγραφών
- Προτιμούμε τη βελτιστοποίηση του μέσου χρόνου πρόσβασης ή του worst-case χρόνου πρόσβασης;
- Αναμενόμενοι τύποι ερωτημάτων:
  - Hashing καλύτερο για point queries
  - Για ερωτήματα εύρους προτιμώνται ταξινομημένα ευρετήρια
- Στην πράξη:
  - PostgreSQL υποστηρίζει ευρετήρια hash με κακή απόδοση
  - Η Oracle υποστηρίζει στατική οργάνωση hash αλλά όχι ευρετήρια
  - Ο SQLServer υποστηρίζει μόνο B^+-trees

Bitmap Ευρετήρια

- Τα Bitmap ευρετήρια είναι ειδικές μορφές ευρετηρίων που σχεδιάστηκαν για αποδοτική εκτέλεση επερωτήσεων σε πολλαπλά κλειδιά.
- Οι εγγραφές σε μια Σχέση αριθμούνται σειριακά ξεκινώντας από το 0.
- Το bitmap είναι απλά ένας πίνακας από τόσα bits όσα ο αριθμός των εγγραφών στη σχέση
- Ένα bitmap ανά διαφορετική τιμή γνωρίσματος
- Εφαρμόζεται καλά σε γνωρίσματα που έχουν ένα σχετικά μικρό αριθμό διαφορετικών τιμών.
  - Π.χ. φύλλο, χώρα, ηλικία, …
  - Π.χ.. επίπεδο-μισθού (ο μισθός έχει χωρισθεί σε ένα αριθμό επιπέδων, όπως 0-9999, 10000-19999, 20000-50000, 50000-άπειρο)

Bitmap Ευρετήρια (συνέχεια)

- Στην απλούστερη μορφή, το bitmap index σε κάποιο γνώρισμα έχει ένα bitmap για κάθε τιμή του γνωρίσματος
  - Το Bitmap έχει τόσα bits όσες είναι οι εγγραφές
  - Στο bitmap για την τιμή v, το bit για μια εγγραφή είναι 1 αν η εγγραφή έχει την τιμή v για το γνώρισμα, και αλλιώς είναι 0

record number	*ID*	*gender*	*income_level*
0	76766	m	L1
1	22222	f	L2
2	12121	f	L1
3	15151	m	L4
4	58583	f	L3

Bitmaps for *gender*

m	10010
f	01101

Bitmaps for *income_level*

L1	10100
L2	01000
L3	00001
L4	00010
L5	00000

Bitmap Ευρετήρια (συνέχεια)

- Τα Bitmap indices δεν είναι καλά για επερωτήσεις σε ένα γνώρισμα
- Οι επερωτήσεις απαντώνται με πράξεις μεταξύ των bitmaps
  - Intersection (and)
  - Union (or)
  - Complementation (not)
- Κάθε πράξη παίρνει δύο bitmaps ιδίου μεγέθους και εφαρμόζει την πράξη στα αντίστοιχα bits για το bitmap του αποτελέσματος
  - Π.χ. 100110 AND 110011 = 100010
    100110 OR 110011 = 110111
    NOT 100110 = 011001
  - ΑΝΔΡΕΣ με ΜΙΣΘΟ ΕΠΙΠΕΔΟΥ L1: 10010 AND 10100 = 10000
    - » Μετά γίνεται πρόσβαση στις πραγματικές εγγραφές!
    - » Η εύρεση του αριθμού των εγγραφών είναι ακόμα πιο γρήγορη

Bitmap Ευρετήρια (συνέχεια)

- Τα Bitmap indices καταλαμβάνουν σχετικά μικρό χώρο
  - Π.χ.. Αν μια εγγραφή είναι 100 bytes, ο χώρος για ένα απλό bitmap είναι 1/800 του χώρου που χρειάζεται για τη Σχέση.
- Η Διαγραφή εγγραφών χρειάζεται ειδική φροντίδα
  - Ένα bitmap «παρουσίας» είναι σύνηθες
  - Χρειάζεται και για άρνηση
    - » not(*A=v*): *(NOT bitmap-A-v) AND ExistenceBitmap*
- Χρειάζονται bitmaps για όλες τις τιμές, ακόμη και τα null
  - Για σωστή χρήση της SQL σημασιολογίας, NOT(*A=v*):
    - » τομή του ανωτέρω αποτελέσματος με (NOT *bitmap-A-Null*)
- Bitmaps μπορεί να χρησιμοποιηθούν εναλλακτικά τωνTuple-ID στα φύλλα των B^+-trees, για τιμές που έχουν ένα μεγάλο αριθμό από matching εγγραφές (συμπίεση)

Ορισμός Ευρετηρίων σε SQL

- Δημιούργησε ένα Ευρετήριο

  create index <index-name> **on** <relation-name>
  (<attribute-list>)

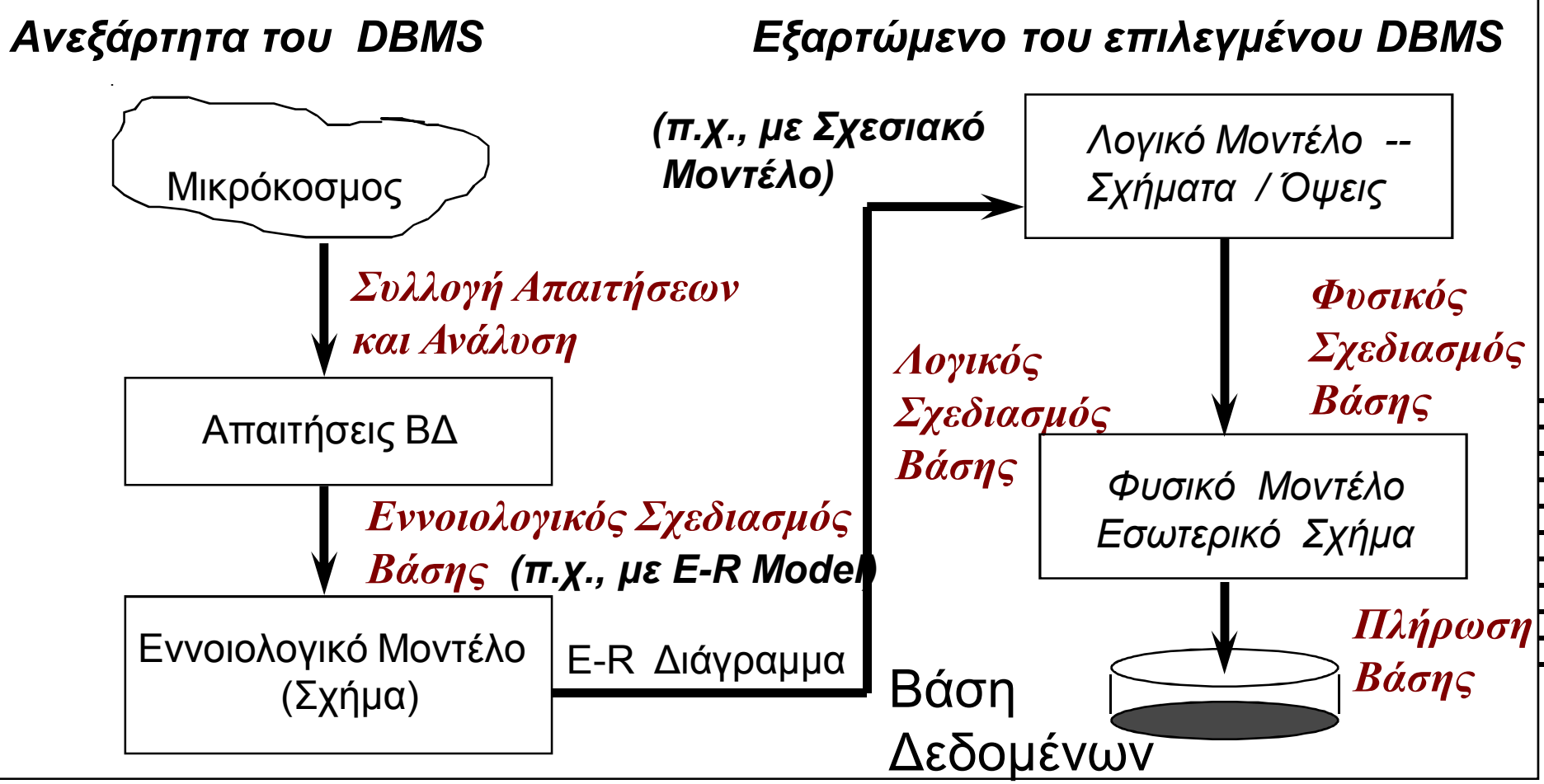
  Π.χ.: **create index** *b-index* **on** *branch(branch-name)*

- Χρησιμοποιείται το **create unique index** ώστε να επιβληθεί η συνθήκη ότι το κλειδί αναζήτησης είναι ένα υποψήφιο κλειδί.

- Διάγραψε ένα ευρετήριο

  drop index <index-name>

Πλήρης Διαδικασία Ανάπτυξης ΒΔ



Επισκόπηση του Φυσικού Σχεδιασμού

- Μετά τον ER σχεδιασμό και τον Λογικό σχεδιασμό (Σχεσιακό μοντέλο), έχουμε τα *εννοιολογικό* και *λογικό (με τις όψεις)* σχήματα για τη Βάση Δεδομένων.
- Το επόμενο βήμα είναι ***ο Φυσικός Σχεδιασμός***, δηλαδή η επιλογή των δομών αποθήκευσης των σχέσεων, η επιλογή των ευρετηρίων, οι αποφάσεις για συστάδες - γενικά ***ότι είναι απαραίτητο για να επιτευχθούν οι προσδοκώμενες Επιδόσεις χρήσης της ΒΔ.***
- Η υλοποίηση μιας (φυσικής) Σχεσιακής Βάσης Δεδομένων περιλαμβάνει τη δημιουργία ΚΑΤΑΛΟΓΩΝ ΣΥΣΤΗΜΑΤΟΣ (directory system tables)

ΚΑΤΑΛΟΓΟΙ ΣΥΣΤΗΜΑΤΟΣ

- Για κάθε Σχέση (Relation):
  - Όνομα, Όνομα Αρχείου, Δομή Αρχείου (π.χ., Αρχείο Σωρού)
  - Όνομα Γνωρίσματος και Τύπος, για κάθε Γνώρισμα
  - Όνομα Ευρετηρίου, για κάθε Ευρετήριο
  - Περιορισμοί Ακεραιότητας
- Για κάθε Ευρετήριο:
  - Δομή (π.χ. B+ δέντρο) και πεδία για αναζήτηση
- Για κάθε Όψη (view):
  - Όνομα Όψης και Ορισμός αυτής
- Επιπλέον, στατιστικά στοιχεία χρήσης, δικαιοδοσίες, μέγεθος ενδιάμεσης μνήμης, κλπ.

Οι Κατάλογοι σε ένα Σχεσιακό Σύστημα αποθηκεύονται και οι ίδιο σαν Σχέσεις

Υλοποίηση Σχέσεων

- **(α) Κάθε Σχέση υλοποιείται με ένα ξεχωριστό αρχείο**
  - Για μικρές σχέσεις, ένας Σωρός - ***heap*** ίσως αρκεί
  - Για μεγαλύτερες Σχέσεις, ***ISAM, B-tree,*** ή ***Hashing***
  - Ορισμός δευτερευόντων ευρετηρίων σε γνωρίσματα

    π.χ.με τις εντολές:

    modify R **to isam on** A_1

    index on R **is S(**A_1)
- **(β) Σχέσεις υλοποιούνται όπως στο DBTG-Δίκτυο**
  - Αποθήκευσε **σχετιζόμενες πλειάδες (από διαφορετικές Σχέσεις) μαζί**
  - Χρησιμοποίησε **δομές με πολλές λίστες** (π.χ., συστάδες)

Φυσικός Σχεδιασμός

- Για να κάνουμε όσο το δυνατόν καλύτερο τον Φυσικό Σχεδιασμό, πρέπει να :

- Κατανοήσουμε το Φόρτο Εργασίας (*workload*)
  - Ποια είναι τα πιο σημαντικά ***queries*** και πόσο συχνά εμφανίζονται.
  - Ποια είναι τα πιο σημαντικά ***updates*** και πόσο συχνά εμφανίζονται.
  - Ποια είναι η ***επιθυμητή επίδοση*** για την εκτέλεση αυτών των queries και updates.

Η κατανόηση του φόρτου εργασίας

- Για κάθε Ερώτηση (query) στο workload:
  - Σε ποιες σχέσεις έχει πρόσβαση?
  - Ποια Γνωρίσματα ανακαλεί?
  - Ποια Γνωρίσματα υπεισέρχονται στις συνθήκες για selection/join? Πόσο επιλεκτικές είναι αυτές οι συνθήκες?
- Για κάθε Ενημέρωση (insert / delete/ update) στο workload:
  - Ο τύπος της ενημέρωσης (INSERT/DELETE/UPDATE), και τα Γνωρίσματα που θα επηρεασθούν

Αποφάσεις που Απαιτούνται

- Τι ευρετήρια πρέπει να δημιουργηθούν?
  - Ποιες σχέσεις πρέπει να έχουν ευρετήρια? Ποια γνωρίσματα χρησιμοποιούνται για αναζήτηση? Πρέπει να ορίσουμε πολλαπλά ευρετήρια?
- Για κάθε ευρετήριο, τι είδους ευρετήριο πρέπει να είναι?
  - Συστάδες? Δέντρο / Κατακερματισμός? Δυναμικό / Στατικό? Πυκνό / Μη-πυκνό?
- Χρειάζονται αλλαγές και στο εννοιολογικό / λογικό Σχήμα?
  - Διαφορετικό κανονικοποιημένο Σχήμα?
  - *Denormalization (μήπως χρειάζεται από-Κανονικοποίηση?)*
  - Όψεις, Επανάληψη Δεδομένων (replication) …

ΕΠΙΛΟΓΗ ΕΥΡΕΤΗΡΙΩΝ

- ***Προσέγγιση:*** Θεώρησε τα πιο σημαντικά queries στη σειρά. Θεώρησε την καλύτερη εκτέλεση (σχέδιο) με τα υπάρχοντα ευρετήρια, και δες αν υπάρχει ακόμη καλύτερη εκτέλεση με ένα επιπλέον ευρετήριο. Αν είναι έτσι, ***δημιούργησέ το***
- Πριν δημιουργήσουμε ένα ευρετήριο, πρέπει να συνυπολογίσουμε και την επίδρασή του σε ενημερώσεις του φορτίου εργασίας!
  - Η εξισορρόπηση είναι ότι ένα ευρετήριο κάνει τις ερωτήσεις ΠΙΟ ΓΡΗΓΟΡΕΣ και τις ενημερώσεις ΠΙΟ ΑΡΓΕΣ
  - Επιπλέον, απαιτεί και χώρο στον Δίσκο

Θέματα για Επιλογή των Ευρετηρίων

- Γνωρίσματα στο WHERE clause είναι υποψήφια για ευρετηρίαση
  - Συνθήκη με ακριβές ταίριασμα (ισότητα) μας οδηγεί σε ευρετήριο κατακερματισμού (hash index.)
  - Ερωτήσεις διακύμανσης μας οδηγούν σε tree index.
    - » Το ευρετήριο συστάδων είναι ιδιαίτερα αποδοτικό για τέτοιου είδους ερωτήσεις – ειδικά αν έχουμε πολλές ίσες τιμές.
- Προσπαθούμε πάντα να επιλέξουμε ευρετήρια που εξυπηρετούν όσα το δυνατό περισσότερα queries.
- Μια και μόνο μία συστάδα-ευρετήριο μπορεί να υπάρχει ανά Σχέση, διάλεξε την MONO αν υπάρχει σημαντικό query που επωφελείται.

Θέματα για Επιλογή των Ευρετηρίων (2)

- Ευρετήρια για πολλαπλά γνωρίσματα δημιουργούνται όταν η WHERE clause περιέχει πολλές συνθήκες
  - Αν υπάρχουν επιλογές διακύμανσης, πρέπει να προσεχθεί η σειρά των γνωρισμάτων
- Όταν υπάρχει συνθήκη συνένωσης:
  - Ανάλογα με τη μέθοδο υλοποίησης της συνένωσης που υποστηρίζεται από το Σύστημα, για παράδειγμα,
    - » Το ευρετήριο μπορεί να είναι κατακερματισμού αν η μέθοδος υλοποίησης συνένωσης είναι nested loop
    - » *Clustered* B+ Δέντρο σε γνωρίσματα συνένωσης είναι καλά για Sort-Merge μέθοδο συνένωσης, κλπ.

Παραδείγματα

- Hash ευρετήριο στο ***D.dname*** υποστηρίζει επιλογή για 'Toy'
  - Αν υπάρχει αυτό, τότε δεν χρειάζεται ευρετήριο στο ***D.dno***
- Hash ευρετήριο στο ***E.dno*** μας επιτρέπει πρόσβαση σε (inner) Emp πλειάδες για κάθε επιλογή μιας (outer) Dept πλειάδας
- Τι θα γινόταν αν το WHERE περιελάμβανε: `` ... AND E.age=25'' ?
  - Ανάκληση των Emp πλειάδων με index στο *E.age*, μετά, συνένωση με Dept πλειάδες (βάσει της επιλογής του *dname*).
  - Άρα, αν το *E.age* index ήδη υπάρχει, αυτό το query δεν δίνει πολλά κίνητρα για την πρόσθεση του *E.dno* index.

```
SELECT  E.ename, D.mgr
FROM  Emp E, Dept D
WHERE  D.dname='Toy' AND E.dno=D.dno
```

Παραδείγματα Clustering

B+ tree index στο E.age μπορεί να χρησιμοποιηθεί για την ανάκληση των πλειάδων

```
SELECT  E.dno
FROM    Emp E
WHERE   E.age>40
```

Θεωρήστε το GROUP BY query.

- Αν για πολλές πλειάδες το *E.age* > 10, η χρήση του *E.age* index και μετά η ταξινόμηση του αποτελέσματος είναι αργή - Clustered *E.dno* index ίσως είναι καλύτερη

```
SELECT E.dno, COUNT (*)
FROM   Emp E
WHERE E.age> 10
GROUP BY E.dno
```

Ερωτήσεις με Ισότητα

- Clustering στο *E.hobby* βοηθά πολύ!

```
SELECT  E.dno
FROM  Emp E
WHERE  E.hobby=Stamps
```

ΣΥΝΟΨΗ

- Η Ανάπτυξη μιας Βάσης Δεδομένων περιλαμβάνει πολλές φάσεις: *ανάλυση απαιτήσεων, εννοιολογικό σχεδιασμό, λογικό σχεδιασμό, φυσικό σχεδιασμό* και *tuning (ρύθμιση).*
  - Εν γένει, πρέπει να πηγαίνουμε μπρος-πίσω από φάση σε φάση για τον βέλτιστο σχεδιασμό, και αποφάσεις σε κάποια φάση επηρεάζουν τις δυνατότητες στις άλλες φάσεις.
- Κατανοώντας τον τύπο του *φόρτου εργασίας* για την εφαρμογή και τις προσδοκώμενες επιδόσεις, είναι σημαντικό ***προαπαιτούμενο*** για τον καλύτερο φυσικό σχεδιασμό
  - Ποια είναι τα πλέον συχνά /σημαντικά queries? Ποια γνωρίσματα / σχέσεις εμπλέκονται? κλπ.

ΣΥΝΟΨΗ (2)

- Τα ευρετήρια χρησιμοποιούνται για την ταχύτερη εκτέλεση των πράξεων
  - Τα ευρετήρια πρέπει επίσης να ενημερώνονται στις Ενημερώσεις του Αρχείου.
  - Επίλεξε ευρετήρια για να εξυπηρετηθούν όσες δυνατόν περισσότερες Σχέσεις / Αρχεία.
  - Η δημιουργία Συστάδας (Cluster) αποτελεί πολύ σημαντική απόφαση: Μια και ΜΟΝΟ ένα Γνώρισμα ανά Σχέση μπορεί να είναι clustered!.
- Τα στατικά ευρετήρια πρέπει περιοδικά να ξανά-δημιουργούνται
- Τα Στατιστικά στοιχεία στους Καταλόγους Συστήματος πρέπει περιοδικά να ανανεώνονται / ενημερώνονται