

# CS 245: Database System Principles

## Notes 5: Hashing and More

Hector Garcia-Molina

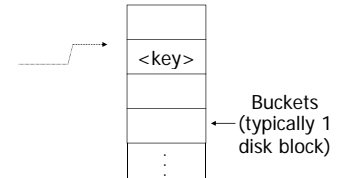
CS 245

Notes 5

1

### Hashing

key  $\rightarrow$  h(key)



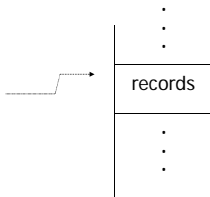
CS 245

Notes 5

2

### Two alternatives

(1) key  $\rightarrow$  h(key)



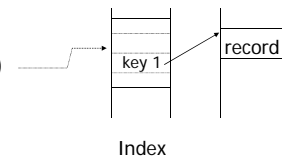
CS 245

Notes 5

3

### Two alternatives

(2) key  $\rightarrow$  h(key)



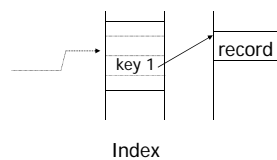
CS 245

Notes 5

4

### Two alternatives

(2) key  $\rightarrow$  h(key)



- Alt (2) for "secondary" search key

CS 245

Notes 5

5

### Example hash function

- Key = 'x<sub>1</sub> x<sub>2</sub> ... x<sub>n</sub>' *n* byte character string
- Have *b* buckets
- h: add x<sub>1</sub> + x<sub>2</sub> + ... + x<sub>n</sub>
  - compute sum modulo *b*

CS 245

Notes 5

6

- ☒ This may not be best function ...
- ☒ Read Knuth Vol. 3 if you really need to select a good function.

CS 245

Notes 5

7

- ☒ This may not be best function ...
- ☒ Read Knuth Vol. 3 if you really need to select a good function.

Good hash function: ☞ Expected number of keys/bucket is the same for all buckets

CS 245

Notes 5

8

Within a bucket:

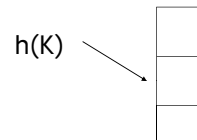
- Do we keep keys sorted?
- Yes, if CPU time critical & Inserts/Deletes not too frequent

CS 245

Notes 5

9

Next: example to illustrate inserts, overflows, deletes



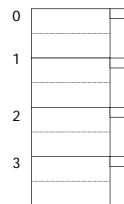
CS 245

Notes 5

10

EXAMPLE 2 records/bucket

INSERT:  
 $h(a) = 1$   
 $h(b) = 2$   
 $h(c) = 1$   
 $h(d) = 0$



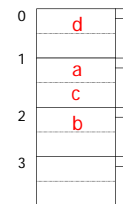
CS 245

Notes 5

11

EXAMPLE 2 records/bucket

INSERT:  
 $h(a) = 1$   
 $h(b) = 2$   
 $h(c) = 1$   
 $h(d) = 0$   
 $h(e) = 1$



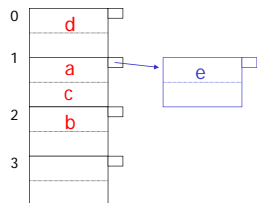
CS 245

Notes 5

12

EXAMPLE 2 records/bucket

INSERT:  
 h(a) = 1  
 h(b) = 2  
 h(c) = 1  
 h(d) = 0  
 h(e) = 1



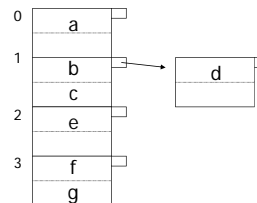
CS 245

Notes 5

13

EXAMPLE: deletion

Delete:  
 e  
 f



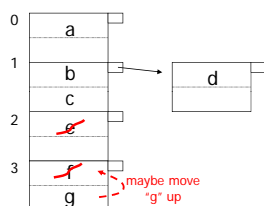
CS 245

Notes 5

14

EXAMPLE: deletion

Delete:  
 e  
 f  
 c



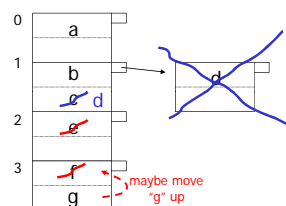
CS 245

Notes 5

15

EXAMPLE: deletion

Delete:  
 e  
 f  
 c



CS 245

Notes 5

16

Rule of thumb:

- Try to keep space utilization between 50% and 80%

$$\text{Utilization} = \frac{\# \text{ keys used}}{\text{total } \# \text{ keys that fit}}$$

CS 245

Notes 5

17

Rule of thumb:

- Try to keep space utilization between 50% and 80%

$$\text{Utilization} = \frac{\# \text{ keys used}}{\text{total } \# \text{ keys that fit}}$$

- If < 50%, wasting space
- If > 80%, overflows significant
  - ↳ depends on how good hash function is & on # keys/bucket

CS 245

Notes 5

18

### How do we cope with growth?

- Overflows and reorganizations
- Dynamic hashing

CS 245

Notes 5

19

### How do we cope with growth?

- Overflows and reorganizations
- Dynamic hashing
  - Extensible
  - Linear

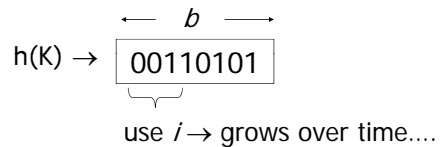
CS 245

Notes 5

20

### Extensible hashing: two ideas

(a) Use  $i$  of  $b$  bits output by hash function

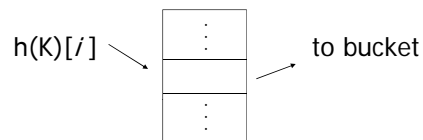


CS 245

Notes 5

21

(b) Use directory

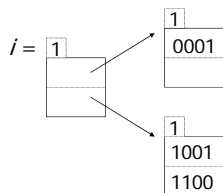


CS 245

Notes 5

22

### Example: $h(k)$ is 4 bits; 2 keys/bucket



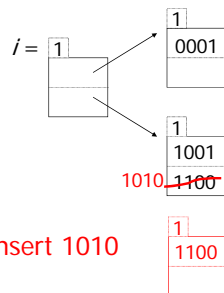
Insert 1010

CS 245

Notes 5

23

### Example: $h(k)$ is 4 bits; 2 keys/bucket

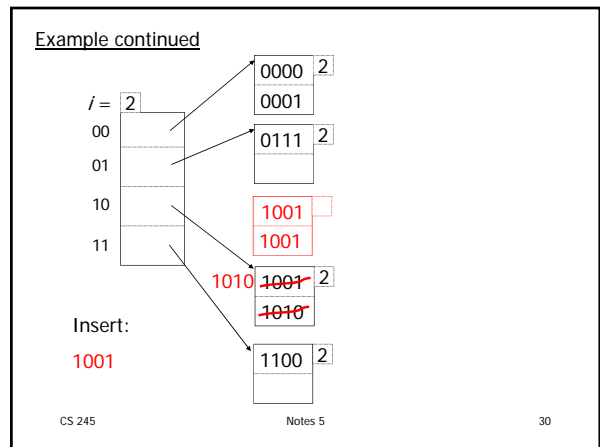
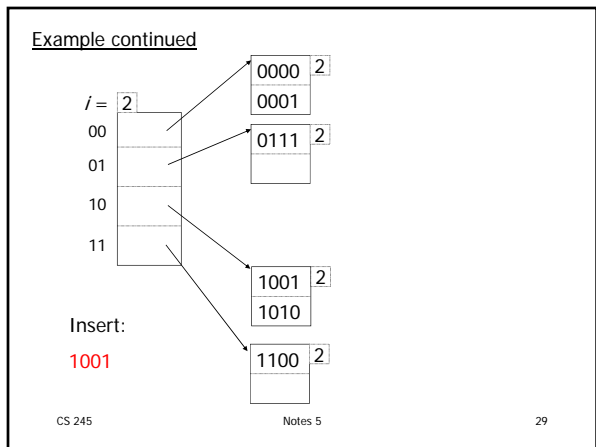
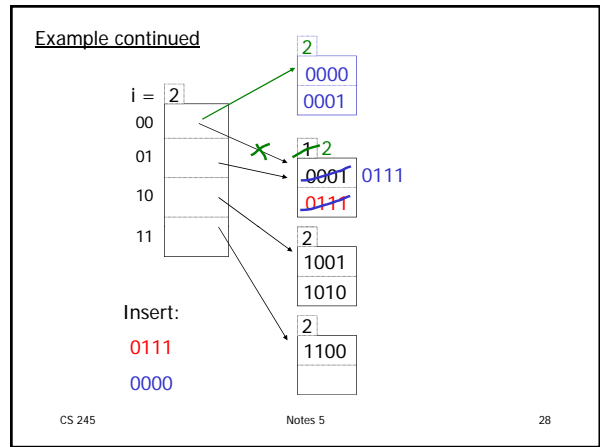
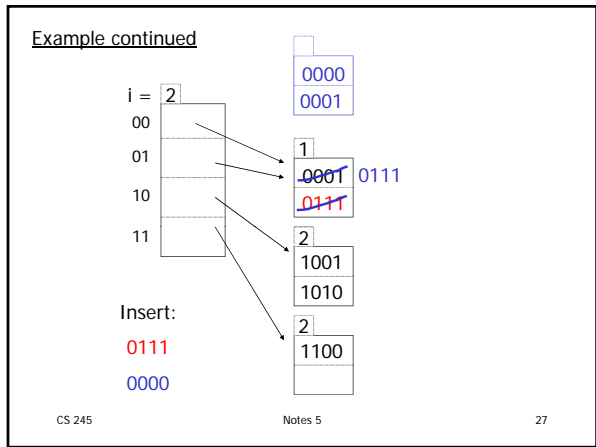
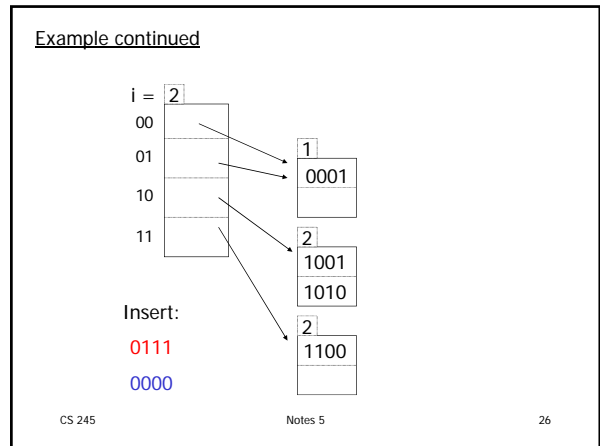
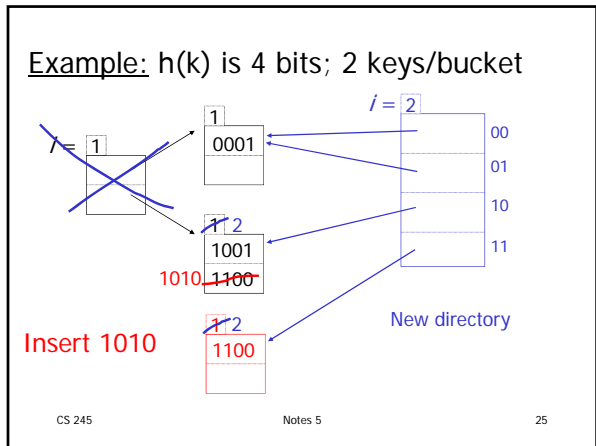


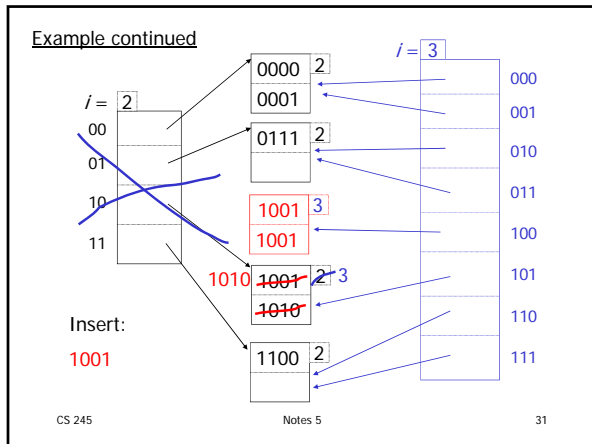
Insert 1010

CS 245

Notes 5

24





Extensible hashing: deletion

- No merging of blocks
- Merge blocks and cut directory if possible (Reverse insert procedure)

CS 245 Notes 5 32

Deletion example:

- Run thru insert example in reverse!

CS 245 Notes 5 33

Note: Still need overflow chains

- Example: many records with duplicate keys

insert 1100      if we split:

CS 245 Notes 5 34

Solution: overflow chains

insert 1100      add overflow block:

CS 245 Notes 5 35

Summary      Extensible hashing

- ⊕ Can handle growing files
  - with less wasted space
  - with no full reorganizations

CS 245 Notes 5 36

## Summary Extensible hashing

- ⊕ Can handle growing files
  - with less wasted space
  - with no full reorganizations
- ⊖ Indirection  
(Not bad if directory in memory)
- ⊖ Directory doubles in size  
(Now it fits, now it does not)

CS 245

Notes 5

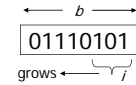
37

## Linear hashing

- Another dynamic hashing scheme

### Two ideas:

- (a) Use  $i$  low order bits of hash



CS 245

Notes 5

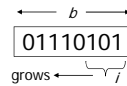
38

## Linear hashing

- Another dynamic hashing scheme

### Two ideas:

- (a) Use  $i$  low order bits of hash



- (b) File grows linearly

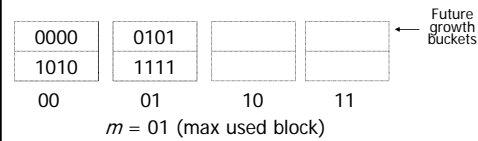


CS 245

Notes 5

39

## Example $b=4$ bits, $i=2$ , 2 keys/bucket

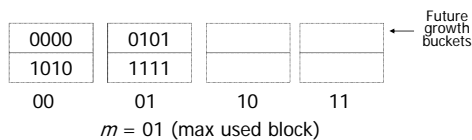


CS 245

Notes 5

40

## Example $b=4$ bits, $i=2$ , 2 keys/bucket



**Rule** If  $h(k)[i] \leq m$ , then  
 look at bucket  $h(k)[i]$   
 else, look at bucket  $h(k)[i] - 2^{i-1}$

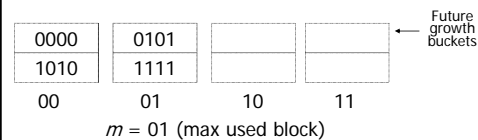
CS 245

Notes 5

41

## Example $b=4$ bits, $i=2$ , 2 keys/bucket

- insert 0101



**Rule** If  $h(k)[i] \leq m$ , then  
 look at bucket  $h(k)[i]$   
 else, look at bucket  $h(k)[i] - 2^{i-1}$

CS 245

Notes 5

42

**Example**  $b=4$  bits,  $i=2$ , 2 keys/bucket

• insert 0101  
• can have overflow chains!

0000 0101  
1010 1111

00 01 10 11

$m = 01$  (max used block)

**Rule** If  $h(k)[i] \leq m$ , then  
look at bucket  $h(k)[i]$   
else, look at bucket  $h(k)[i] - 2^{i-1}$

CS 245 Notes 5 43

**Note**

- In textbook,  $n$  is used instead of  $m$
- $n = m + 1$

0000 0101  
1010 1111

00 01 10 11

$m = 01$  (max used block)

CS 245 Notes 5 44

**Example**  $b=4$  bits,  $i=2$ , 2 keys/bucket

0000 0101  
1010 1111

00 01 10 11

$m = 01$  (max used block)

CS 245 Notes 5 45

**Example**  $b=4$  bits,  $i=2$ , 2 keys/bucket

0000 0101 1010  
1010 1111

00 01 10 11

$m = 01$  (max used block)  
10

CS 245 Notes 5 46

**Example**  $b=4$  bits,  $i=2$ , 2 keys/bucket

• insert 0101

0000 0101 1010  
1010 1111

00 01 10 11

$m = 01$  (max used block)  
10

CS 245 Notes 5 47

**Example**  $b=4$  bits,  $i=2$ , 2 keys/bucket

• insert 0101

0000 0101 1010  
1010 1111

00 01 10 11

$m = 01$  (max used block)  
10  
11

CS 245 Notes 5 48

Example  $b=4$  bits,  $i=2$ , 2 keys/bucket

• insert 0101

0000	0101	1010	1111
<del>1010</del>	<del>0101</del>		

00      01      10      11

$m = 11$  (max used block)

Future growth buckets

CS 245      Notes 5      49

Example Continued: How to grow beyond this?

$i = 2$

0000	0101	1010	1111
	0101		

00      01      10      11      ...

$m = 11$  (max used block)

CS 245      Notes 5      50

Example Continued: How to grow beyond this?

$i = 2^3$

0000	0101	1010	1111		
	0101				

000	001	010	011	...
100	101	110	111	

$m = 11$  (max used block)

CS 245      Notes 5      51

Example Continued: How to grow beyond this?

$i = 2^3$

0000	0101	1010	1111		
	0101				

000	001	010	011	100	...
<del>100</del>	101	110	111		

$m = 11$  (max used block)

100

CS 245      Notes 5      52

Example Continued: How to grow beyond this?

$i = 2^3$

0000	0101	1010	1111		0101
	0101				0101

000	001	010	011	100	101	...
<del>100</del>	<del>101</del>	110	111			

$m = 11$  (max used block)

100

101

CS 245      Notes 5      53

☒ When do we expand file?

- Keep track of:  $\frac{\# \text{ used slots}}{\text{total \# of slots}} = U$

CS 245      Notes 5      54

☒ When do we expand file?

- Keep track of:  $\frac{\# \text{ used slots}}{\text{total \# of slots}} = U$
- If  $U > \text{threshold}$  then increase  $m$   
(and maybe  $i$ )

CS 245

Notes 5

55

Summary Linear Hashing

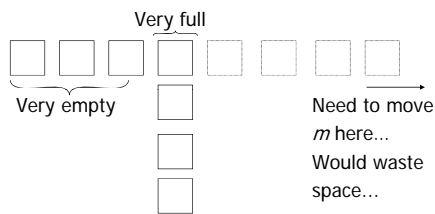
- ⊕ Can handle growing files
  - with less wasted space
  - with no full reorganizations
- ⊕ No indirection like extensible hashing
- ⊖ Can still have overflow chains

CS 245

Notes 5

56

Example: BAD CASE



CS 245

Notes 5

57

Summary

Hashing

- How it works
- Dynamic hashing
  - Extensible
  - Linear

CS 245

Notes 5

58

Next:

- Indexing vs Hashing
- Index definition in SQL
- Multiple key access

CS 245

Notes 5

59

Indexing vs Hashing

- Hashing good for probes given key  
e.g.,  
SELECT ...  
FROM R  
WHERE R.A = 5

CS 245

Notes 5

60

## Indexing vs Hashing

- INDEXING (Including B Trees) good for Range Searches:  
e.g.,  
SELECT  
FROM R  
WHERE R.A > 5

CS 245

Notes 5

61

## Index definition in SQL

- Create index name on rel (attr)
- Create unique index name on rel (attr)  
└─┬─┘ defines candidate key
- Drop INDEX name

CS 245

Notes 5

62

**Note** CANNOT SPECIFY TYPE OF INDEX  
(e.g. B-tree, Hashing, ...)  
OR PARAMETERS  
(e.g. Load Factor, Size of Hash,...)

... at least in SQL...

CS 245

Notes 5

63

**Note** ATTRIBUTE LIST ⇒ MULTIKEY INDEX  
(next)

e.g., CREATE INDEX foo ON R(A,B,C)

CS 245

Notes 5

64

## Multi-key Index

Motivation: Find records where  
DEPT = "Toy" AND SAL > 50k

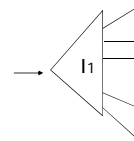
CS 245

Notes 5

65

## Strategy I:

- Use one index, say Dept.
- Get all Dept = "Toy" records  
and check their salary

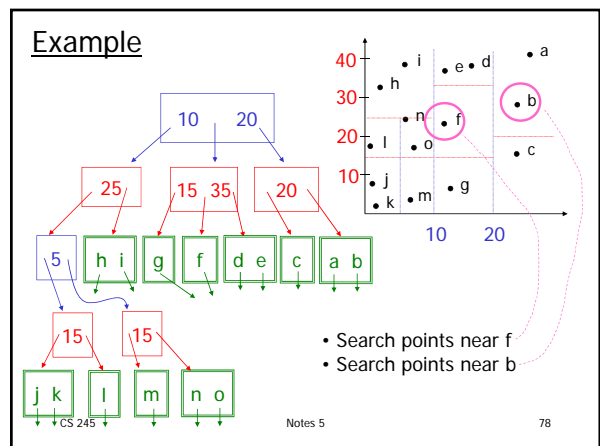
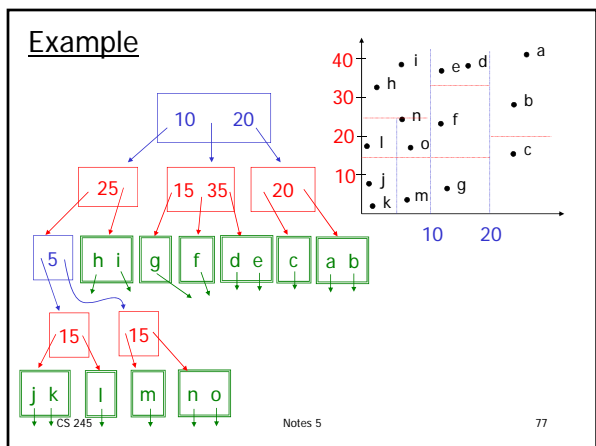
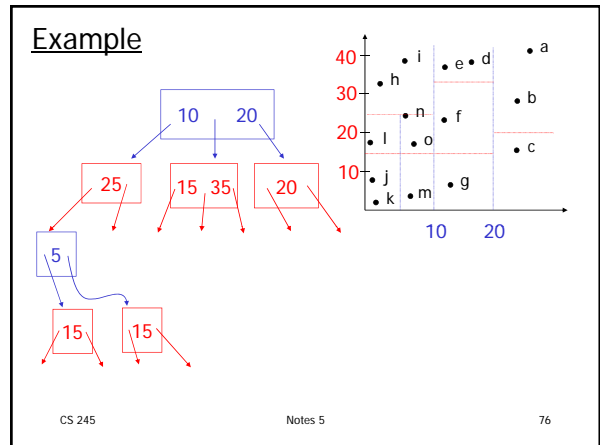
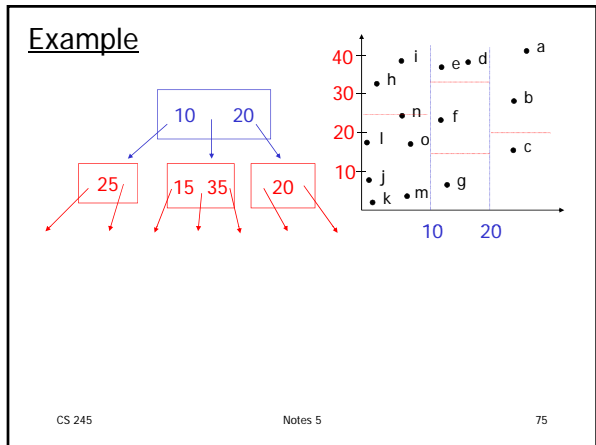
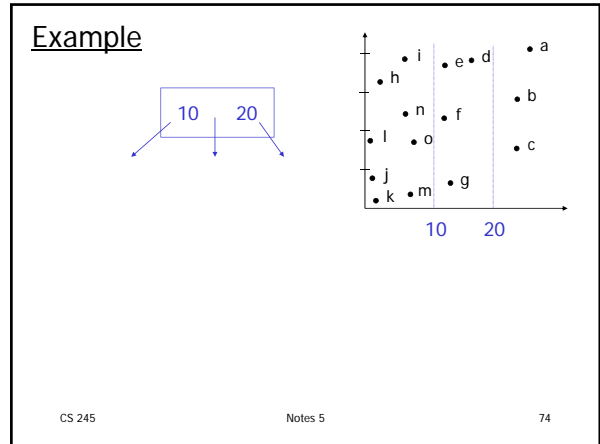
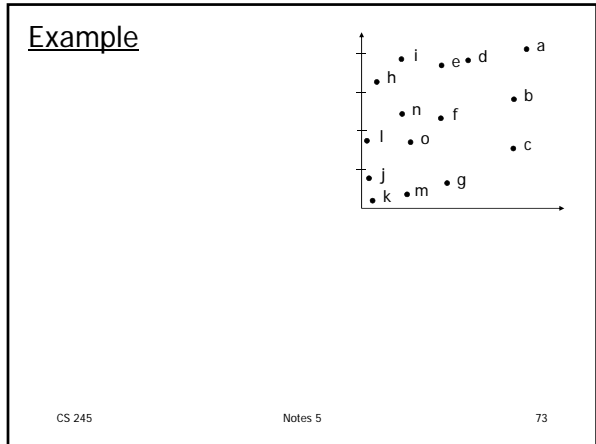


CS 245

Notes 5

66





### Queries

- Find points with  $Y_i > 20$
- Find points with  $X_i < 5$
- Find points "close" to  $i = \langle 12, 38 \rangle$
- Find points "close" to  $b = \langle 7, 24 \rangle$

CS 245

Notes 5

79

- Many types of geographic index structures have been suggested

- kd-Trees (very similar to what we described here)
- Quad Trees
- R Trees
- ...

CS 245

Notes 5

80

### Two more types of multi key indexes

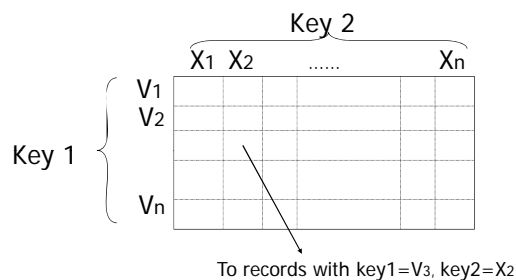
- Grid
- Partitioned hash

CS 245

Notes 5

81

### Grid Index



CS 245

Notes 5

82

### CLAIM

- Can quickly find records with
  - key 1 =  $V_i \wedge$  Key 2 =  $X_j$
  - key 1 =  $V_i$
  - key 2 =  $X_j$

CS 245

Notes 5

83

### CLAIM

- Can quickly find records with
  - key 1 =  $V_i \wedge$  Key 2 =  $X_j$
  - key 1 =  $V_i$
  - key 2 =  $X_j$
- And also ranges....
  - E.g., key 1  $\geq V_i \wedge$  key 2  $< X_j$

CS 245

Notes 5

84

☒ But there is a catch with Grid Indexes!

- How is Grid Index stored on disk?

Like Array...

V1	V2	V3
X1	X1	X1
X2	X2	X2
X3	X3	X3
X4	X4	X4

CS 245 Notes 5 85

☒ But there is a catch with Grid Indexes!

- How is Grid Index stored on disk?

Like Array...

V1	V2	V3
X1	X1	X1
X2	X2	X2
X3	X3	X3
X4	X4	X4

**Problem:**

- Need regularity so we can compute position of  $\langle V_i, X_j \rangle$  entry

CS 245 Notes 5 86

Solution: Use Indirection

	X1	X2	X3	
V1				...
V2				...
V3				...
V4				...

Buckets

\*Grid only contains pointers to buckets

Buckets

CS 245 Notes 5 87

With indirection:

- Grid can be regular without wasting space
- We do have price of indirection

CS 245 Notes 5 88

Can also index grid on value ranges

Salary		→	Grid		
0-20K	1				
20K-50K	2				
50K-∞	3				

Linear Scale →

1	2	3
Toy	Sales	Personnel

CS 245 Notes 5 89

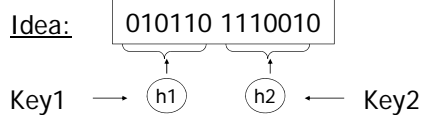
Grid files

- ⊕ Good for multiple-key search
- ⊖ Space, management overhead (nothing is free)
- ⊖ Need partitioning ranges that evenly split keys

CS 245 Notes 5 90

## Partitioned hash function

Idea:



CS 245

Notes 5

91

EX:

h1(toy)	=0	000	
h1(sales)	=1	001	
h1(art)	=1	010	
.		011	
h2(10k)	=01	100	
h2(20k)	=11	101	
h2(30k)	=01	110	
h2(40k)	=00	111	
.			

Insert → <Fred, toy, 10k>, <Joe, sales, 10k>  
<Sally, art, 30k>

CS 245

Notes 5

92

EX:

h1(toy)	=0	000	
h1(sales)	=1	001	<Fred>
h1(art)	=1	010	
.		011	
h2(10k)	=01	100	
h2(20k)	=11	101	<Joe><Sally>
h2(30k)	=01	110	
h2(40k)	=00	111	
.			

Insert → <Fred, toy, 10k>, <Joe, sales, 10k>  
<Sally, art, 30k>

CS 245

Notes 5

93

h1(toy)	=0	000	<Fred>
h1(sales)	=1	001	<Joe><Jan>
h1(art)	=1	010	<Mary>
.		011	
h2(10k)	=01	100	<Sally>
h2(20k)	=11	101	
h2(30k)	=01	110	<Tom><Bill>
h2(40k)	=00	111	<Andy>
.			

• Find Emp. with Dept. = Sales  $\wedge$  Sal=40k

CS 245

Notes 5

94

h1(toy)	=0	000	<Fred>
h1(sales)	=1	001	<Joe><Jan>
h1(art)	=1	010	<Mary>
.		011	
h2(10k)	=01	100	<Sally>
h2(20k)	=11	101	
h2(30k)	=01	110	<Tom><Bill>
h2(40k)	=00	111	<Andy>
.			

• Find Emp. with Dept. = Sales  $\wedge$  Sal=40k

CS 245

Notes 5

95

h1(toy)	=0	000	<Fred>
h1(sales)	=1	001	<Joe><Jan>
h1(art)	=1	010	<Mary>
.		011	
h2(10k)	=01	100	<Sally>
h2(20k)	=11	101	
h2(30k)	=01	110	<Tom><Bill>
h2(40k)	=00	111	<Andy>
.			

• Find Emp. with Sal=30k

CS 245

Notes 5

96

h1(toy)	=0	000	<Fred>
h1(sales)	=1	001	<Joe><Jan>
h1(art)	=1	010	<Mary>
.	.	011	
h2(10k)	=01	100	<Sally>
h2(20k)	=11	101	
h2(30k)	=01	110	<Tom><Bill>
h2(40k)	=00	111	<Andy>
.	.		

• Find Emp. with Sal=30k look here

CS 245 Notes 5 97

h1(toy)	=0	000	<Fred>
h1(sales)	=1	001	<Joe><Jan>
h1(art)	=1	010	<Mary>
.	.	011	
h2(10k)	=01	100	<Sally>
h2(20k)	=11	101	
h2(30k)	=01	110	<Tom><Bill>
h2(40k)	=00	111	<Andy>
.	.		

• Find Emp. with Dept. = Sales

CS 245 Notes 5 98

h1(toy)	=0	000	<Fred>
h1(sales)	=1	001	<Joe><Jan>
h1(art)	=1	010	<Mary>
.	.	011	
h2(10k)	=01	100	<Sally>
h2(20k)	=11	101	
h2(30k)	=01	110	<Tom><Bill>
h2(40k)	=00	111	<Andy>
.	.		

• Find Emp. with Dept. = Sales look here

CS 245 Notes 5 99

Summary

Post hashing discussion:

- Indexing vs. Hashing
- SQL Index Definition
- Multiple Key Access
  - Multi Key Index
    - Variations: Grid, Geo Data
  - Partitioned Hash

CS 245 Notes 5 100


### Reading Chapter 5

- Skim the following sections:
  - Sections 14.3.6, 14.3.7, 14.3.8  
[Second Ed: 14.6.6, 14.6.7, 14.6.8]
  - Sections 14.4.2, 14.4.3, 14.4.4  
[Second Ed: 14.7.2, 14.7.3, 14.7.4]
- Read the rest

CS 245 Notes 5 101

### The BIG picture....

- Chapters 11 & 12 [13]: Storage, records, blocks...
- Chapters 13 & 14 [14]: Access Mechanisms
  - Indexes
  - B trees
  - Hashing
  - Multi key
- Chapters 15 & 16 [15, 16]: Query Processing



CS 245 Notes 5 102