Lecture 22: Hash Tables and Query Execution

Wednesday, Nov. 21, 2001

1

Outline

- Hash tables (4.4)
- Query execution (6.1-6.5)

2

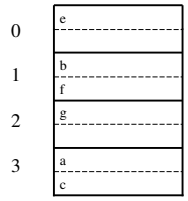
Hash Tables

- Secondary storage hash tables are much like main memory ones
- Recall basics:
  - There are n *buckets*
  - A hash function f(k) maps a key k to {0, 1, …, n-1}
  - Store in bucket f(k) a pointer to record with key k
- Secondary storage: bucket = block, use overflow blocks when needed

3

Hash Table Example

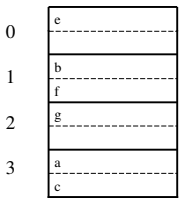
- Assume 1 bucket (block) stores 2 keys + pointers
- h(e)=0
- h(b)=h(f)=1
- h(g)=2
- h(a)=h(c)=3



4

Searching in a Hash Table

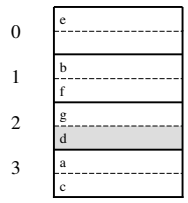
- Search for a:
- Compute h(a)=3
- Read bucket 3
- 1 disk access



5

Insertion in Hash Table

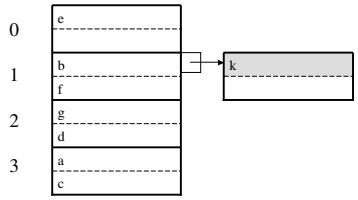
- Place in right bucket, if space
- E.g. h(d)=2



6

Insertion in Hash Table

- Create overflow block, if no space
- E.g. h(k)=1



- More over-flow blocks may be needed

7

Hash Table Performance

- Excellent, if no overflow blocks
- Degrades considerably when number of keys exceeds the number of buckets (I.e. many overflow blocks).

8

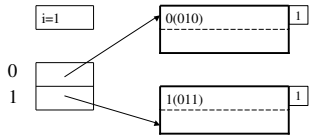
Extensible Hash Table

- Allows has table to grow, to avoid performance degradation
- Assume a hash function h that returns numbers in $\{0, \ldots, 2^k - 1\}$
- Start with $n = 2^i << 2^k$, only look at first i most significant bits

9

Extensible Hash Table

- E.g. i=1, n=2^i=2, k=4

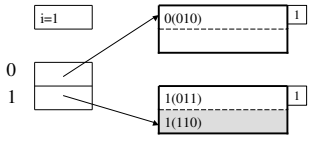


- Note: we only look at the first bit (0 or 1)

10

Insertion in Extensible Hash Table

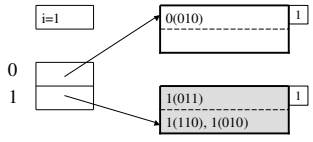
- Insert 1110



11

Insertion in Extensible Hash Table

- Now insert 1010

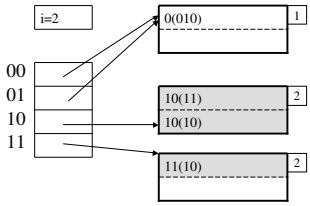


- Need to extend table, split blocks
- i becomes 2

12

Insertion in Extensible Hash Table

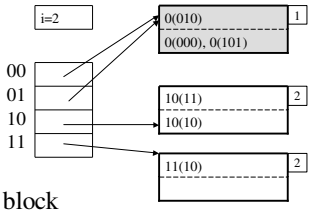
- Now insert 1110



13

Insertion in Extensible Hash Table

- Now insert 0000, then 0101

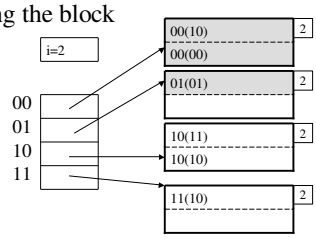


- Need to split block

14

Insertion in Extensible Hash Table

- After splitting the block



15

Performance Extensible Hash Table

- No overflow blocks: access always one read
- BUT:
  - Extensions can be costly and disruptive
  - After an extension table may no longer fit in memory

16

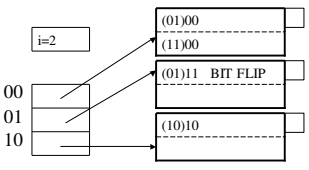
Linear Hash Table

- Idea: extend only one entry at a time
- Problem: n= no longer a power of 2
- Let i be such that $2^i <= n < 2^{i+1}$
- After computing h(k), use last i bits:
  - If last i bits represent a number > n, change msb from 1 to 0 (get a number <= n)

17

Linear Hash Table Example

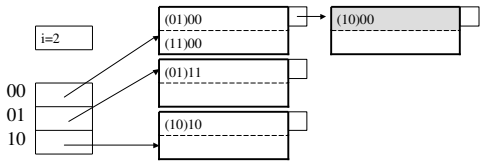
- n=3



18

Linear Hash Table Example

- Insert 1000: overflow blocks…



19

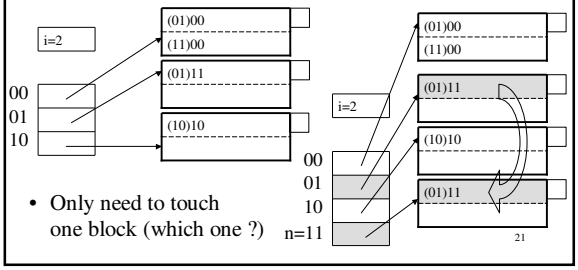
Linear Hash Tables

- Extension: independent on overflow blocks
- Extend n:=n+1 when average number of records per block exceeds (say) 80%

20

Linear Hash Table Extension

- From n=3 to n=4

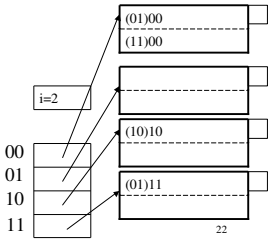


- Only need to touch one block (which one ?)

21

Linear Hash Table Extension

- From n=3 to n=4 finished
- Extension from n=4 to n=5 (new bit)
- Need to touch every single block (why ?)



22

An Algebra for Queries

- Logical operators
  - *what* they do
- Physical operators
  - *how* they do it

23

Logical Operators in the Algebra

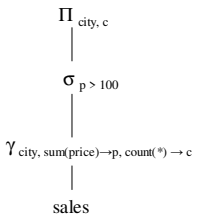
- Union, intersection, difference
- Selection σ
- Projection Π
- Join ⋉
- Duplicate elimination δ
- Grouping γ
- Sorting τ

Relational Algebra

24

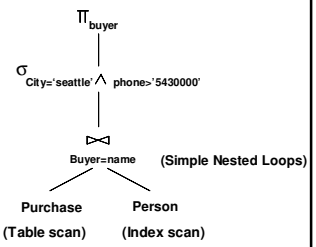
Example

```
SELECT city, count(*)
FROM sales
GROUP BY city
HAVING sum(price) > 100
```



Physical Operators

```
SELECT S.buyer
FROM Purchase P, Person Q
WHERE P.buyer=Q.name AND
      Q.city='seattle' AND
      Q.phone > '5430000'
```



Query Plan:
- logical tree
- implementation choice at every node
- scheduling of operations.

Some operators are from relational algebra, and others (e.g., scan, group) are not.

Scanning Tables

- The table is *clustered* (i.e. blocks consists only of records from this table):
  - Table-scan: if we know where the blocks are
  - Index scan: if we have a sparse index to find the blocks
- The table is *unclustered* (e.g. its records are placed on blocks with other tables)
  - May need one read for each record

Sorting While Scanning

- Sometimes it is useful to have the output sorted
- Three ways to scan it sorted:
  - If there is a primary or secondary index on it, use it during scan
  - If it fits in memory, sort there
  - If not, use multi-way merge sort

Estimating the Cost of Operators

- Very important for the optimizer (next week)
- Parameters for a relation R
  - B(R) = number of blocks holding R
    - Meaningful if R is clustered
  - T(R) = number of tuples in R
    - E.g. may need when R is unclustered
  - V(R,a) = number of distinct values of the attribute a

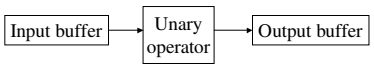
Cost of the Scan Operator

- Clustered relation:
  - Table scan: B(R); to sort: 3B(R)
  - Index scan: B(R); to sort: B(R) or 3B(R)
- Unclustered relation
  - T(R); to sort: T(R) + 2B(R)

One-Pass Algorithms

Selection $\sigma(R)$, projection $\Pi(R)$

- Both are *tuple-at-a-time* algorithms
- Cost: B(R)



31

One-pass Algorithms

Duplicate elimination $\delta(R)$

- Need to keep tuples in memory
- When new tuple arrives, need to compare it with previously seen tuples
- Balanced search tree, or hash table
- Cost: B(R)
- Assumption: $B(\delta(R)) <= M$

32

One-pass Algorithms

Grouping: $\gamma_{city,\ sum(price)}$ (R)

- Need to store all cities in memory
- Also store the sum(price) for each city
- Balanced search tree or hash table
- Cost: B(R)
- Assumption: number of cities fits in memory

33

One-pass Algorithms

Binary operations: $R \cap S$, $R \cup S$, $R - S$

- Assumption: $\min(B(R), B(S)) <= M$
- Scan one table first, then the next, eliminate duplicates
- Cost: B(R)+B(S)

34