ICOM 6005 – Database Management Systems Design

Dr. Manuel Rodríguez-Martínez

Electrical and Computer Engineering Department

Query Evaluation Techniques

- Read :
  - Chapter 12, sec 12.1-12.3
  - Chapter 13
- Purpose:
  - Study different algorithms to execute (evaluate) SQL relational operators
    - Selection
    - Projection
    - Joins
    - Aggregates
    - Etc.

Selections using a B+-tree

- When a selection has a where clause with a predicate of the form *Attr op value*
  - Clustered B+-tree: best strategy to use (if available)
    - Note that hash index is better if op is equality
  - Unclustered B+ tree: depends on SF for the predicate
  - Algorithm:
    - Search tree to find first index entry that points to qualifying tuples (tuples that pass the where condition)
    - Scan data entries to find and retrieve qualifying tuples
- Costs:
  - Clustered: # of visited index pages + # of visited data entries
  - Unclustered: # of visited index pages + # of visited data entries + # of file data page read
    - Worst case: 1 file data page is read for each tuple

Some issues …

- If the B+ tree index is clustered it is very probable that 1 page has all the data records
  - Since # of visited index entries is often 2 - 3 I/Os, total operations can be implemented in 4 I/Os!
- If the index is unclustered, worst case is that a file data page is read for each tuple
  - **Example**: Relation with 1000 tuples and 100 page will cost up to: 3 + 100*1000 = 100,003 I/Os (This can't be good!)
- Typically, DBMS first determines the page ids of the pages to read from a unclustered B+ tree
- Then, sorts the page ids, reads pages in order and fetches the tuples from them
  - This prevents a given page to be read more than once!

Example:

- Q1: Select sid, sname from Students where gap = 4.0
- Q2: Select sname, gpa, sage from Students where age < 20;
- NTuples(Students):10,000, NPages(Students) = 1000
- Indices:
  - **Clustered Hash Index on sid: INKeys = 10,000, INPages= 500**
  - **Clustered B+Tree on age: INIndex= 40, INPages = 500**
- Selectivity factors:
  - **Gpa 4.0 = 10%**
  - **Age < 20 = 40 %**
- What should be the strategy to solve each of the queries above?

General Selection Conditions

- DBMS often deals with where clause in two forms
  - Where clause has not disjunctions
    - Or clauses
  - Where clause has an or condition
- Each predicate in the where clause has a free from
  - Attr op value
  - Attr1 op Attr2
  - Attr1 op func1(…) // embedded function call
  - Func1(Attr1, …., Attrn) // function call is the predicate
- DBMS excel in optimizing where clauses with no disjunctions
  - They drop the ball when an OR appears on where clause

Query with predicates in CNF

- Condition: Atttr1 op value And attr2 op value …
- DBMS estimates the selectivity of predicate + index availability
- The predicate that is the most selective is evaluated first
  - The rest are evaluated based on selectivity
- Alternatively, if an index exist for a subset of conditions
  - Each Condition is evaluated separately
  - Results are then intersected
  - Any remaining predicated are then evaluated

Query with predicate in DNF

- Strategy 1:
  - File scan and evaluate the where clause
  - Cost: Read the entire relation
- Strategy 2
  - Separate each term in where in a separte query
  - Union results
  - Cost: sum of cost of individual operations.
- Example:
  - Select sid, sname from Students where age = 20 OR gpa > 3.50
  - Become the union between:
    - Select sid, sname from Students where age = 20;
    - Select sid, sname from Students where gpq > 3.50;

Evaluation of Projections

- The main issue is duplicate elimination.
  - If no duplication elimination is need, just scan tuples and project attributes.
  - If selection was applied first, simply project tuples being collected from selection operator.
    - Select sid, sname from Students;
    - Select sid from Students where gpa = 4.0;
- Duplicate elimination makes things harder
  - Need to project tuples
  - Remove duplicates from this result set
- Strategies for duplicate elimination
  - Partition via Sorting (using external sorting)
  - Partition via Hashing
  - Indexing on projected attributes

Projections via Sorting

- Strategy:
  1. Scan relation R and project tuples to get desired attributes
  2. Store projected tuples into a temporary relation T
  3. Sort the resulting set of tuples from previous step.
     - Key is the set of all attributes
  4. Scan the sorted set of tuples, compared adjacent tuples, and only keep a copy of repeated tuples (discard others)
- Costs estimates
  - Step 1: NPages(R) I/Os
  - Step 2: NPages(T) I/Os
  - Step 3: O(NlogN), where N = NPages(T)
  - Step 4: NPages(T) I/Os
  - Computational Complexity of algorithm:
    - **O(NlogN), N = Pages(T)**

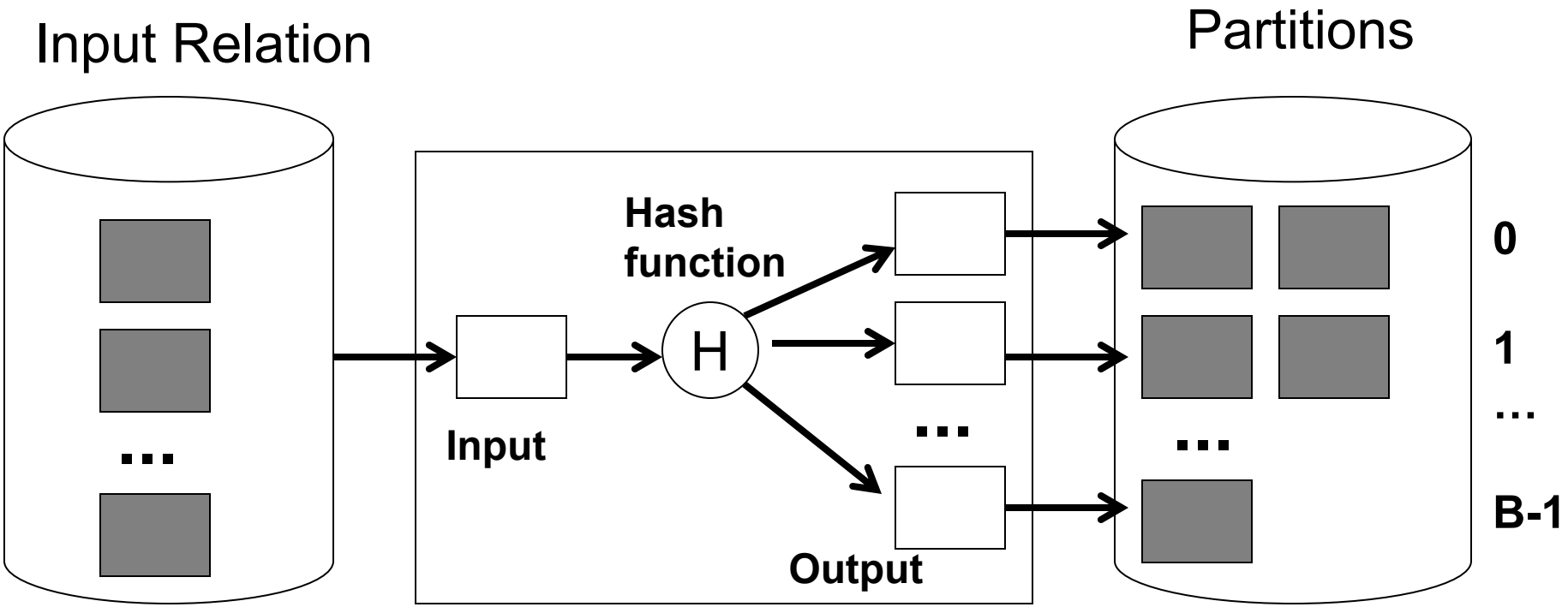
Example: Evaluation via Sorting

- Query: Select distinct R.sid, R.bid from Reserves R;
- Reserves:
  - NTuples(R) = 100,000
  - Tuples size = 100 bytes
  - Page Size = 4096 bytes
  - Buffers for sorting = 6
  - Size of projected tuples: 16 bytes
- How much does it costs to evaluate this projection?
  - Step 1 = $\lceil 100{,}000 / \lfloor 4096/100 \rfloor \rceil$ I/0s = 2,500 I/Os
  - Step 2 = $\lceil 100{,}000 / \lfloor 4096/16 \rfloor \rceil$ I/0s = 391 I/Os
  - Step 3 = $2*391*(\lceil \log_5 \lceil 391/6 \rceil \rceil + 1) = 2*391*4 = 3128$ I/Os
  - Step 4 = 391 I/Os
  - Total = 2500 + 391 + 3128 +391 = 6410 I/Os

Projections via Hashing

- If we have a lot of memory buffers, hashing provides an attractive alternative
  - Modern DBMS servers have Gigabytes of RAM
- Suppose we have B buffers to use for projections
- General idea is as follows:
  - Have B -1 buckets resident on disk
    - **Temporary files on disk**
  - Use 1 buffer to keep tuples read from relation R
  - Use a hash function to hash each tuple to a bucket
  - Use 1 buffer per bucket to keep hashed tuples in memory
    - **Flush page to disk-resident bucket only when page is full**
    - **This forms an in-memory hash table**
  - Remove duplicates at the buckets on disk

Projections via Hashing (scheme)



Build B-1 disk-resident partitions of variables size

Some Issues

- Hash function should distribute tuples uniformly
- This option for projection evaluation is very memory consuming
  - Should only be used if enough buffers are available
- How many buffer is enough?
  - Let B be the number of buffers to use
  - We have B-1 partitions
  - Let M be the number of pages with tuples after projecting table R
  - We have T/B-1 pages in each partition
  - Hash table size will be *(T/B-1) * f* , f is fudge factor to compensate for extra space need
  - Thus,

$$B > \sqrt{f * T}$$