



ΕΡΓΑΣΤΗΡΙΟ #2

ΜΕΡΟΣ Α': Ασκήσεις: Storage and Indexing Overview

1. **(Exercise 8.3)** Consider a relation stored as a randomly ordered file for which the only index is an unclustered index on a field called sal. If you want to retrieve all records with $sal > 20$, is using the index always the best alternative? Explain.

Answer: No. In this case, the index is unclustered, each qualifying data entry could contain an rid that points to a distinct data page, leading to as many data page I/Os as the number of data entries that match the range query. In this situation, using index is actually worse than file scan.

2. **(Exercise 8.9)** What main conclusions can you draw from the discussion of the five basic file organizations discussed in Section 8.4? (Heap, Sorted, Clustered, Unclustered tree index, Unclustered hash index **Book p.283**) Which of the five organizations would you choose for a file where the most frequent operations are as follows?
- Search for records based on a range of field values.
 - Perform inserts and scans, where the order of records does not matter.
 - Search for a record based on a particular field value

File Type	Scan	Equality Search	Range Search	Insert	Delete
Heap	BD	0.5BD	BD	2D	Search + D
Sorted	BD	$D\log_2(B)$	$D\log_2(B) + \text{matching pages}$	Search + BD	Search + BD
Clustered	1.5BD	$D\log_F(1.5B)$	$D\log_F(1.5B) + \text{matching pages}$	Search + D	Search + D
Unclustered tree index	$BD(R + 0.15)$	$D + D\log_F(0.15B)$	$D\log_F(\text{index size}) + D \cdot \text{matching records}$	$3D + D\log_F(\text{index size})$	Search + 2D
Unclustered hash index	$BD(R + 0.125)$	2D	BD	4D	Search + 2D

B = Number of data pages when records are packed onto pages with no wasted space

D = Average time to read or write a disk page

F = Fan-out (tree indexes). Typically at least 100.

Heap Files

- Scan: Cost is BD since we have to retrieve each of B pages with each page taking D time.
- Equality Search: If exactly one record matches the desired equality search then on average we must scan half of the file, assuming record exists in only that part of file. Hence cost is $0.5BD$.
- Range Search: In this entire file must be scanned for matching records. So cost is BD .
- Insert: If records are inserted at the end of page the time taken is fetching the page and writing back the page. So cost is $2D$.
- Delete: Here time taken is searching for relevant record and writing back the page after deleting record from it. So cost is $\text{Search} + D$.

Sorted Files

- Scan: Cost is BD since we have to retrieve each of B pages with each page taking D time.
- Equality Search: If we assume that the equality search is specified on the field by which the file is sorted, then we can search for the record by the help of binary search. Hence cost is $D\log_2(B)$.
- Range Search: It is equality search for all matching records. So cost is $D\log_2(B) + \text{matching pages}$.
- Insert: To insert the record while preserving the sorted order, first we have to search for the correct position in the file, add record and then fetch and rewrite all subsequent pages. So cost is $\text{Search} + BD$.
- Delete: Here we search for record, remove the record from the page, and rewrite the subsequent pages to fill the space created by the record which is deleted. Hence cost is $\text{Search} + BD$.

Clustered Tree Index

- Scan: Here effective number of pages is 1.5 times more than pages in heap files since page occupancy is 67%. So, Cost is $1.5BD$ since we have to retrieve all the pages with each page taking D time.
- Equality Search: If data records are ordered as data entries in some index, then we do F -ary search. So cost is $D\log_F(1.5B)$.
- Range Search: It is equality search for all matching records. So cost is $D\log_F(1.5B) + \text{matching pages}$.
- Insert: Here time required is for searching correct position for record in the page and writing back the page. So cost is $\text{Search} + D$.
- Delete: Similar to insert, first search for page, delete record from it and write back the page. Cost is $\text{Search} + D$.

Unclustered Tree Index

Assumptions: the size of one data entry is 10% the size of one record; also, index pages have $2/3=67\%$ occupancy; therefore, number of index leaf pages is $0.1 \cdot 1.5B = 0.15B$

- Scan: Here each record takes D time to read from a single page. So reading R record from a page takes DR time. Hence total cost for B pages is $BDR + \text{Read index}$.
- Equality Search: If we assume that data index size is one-tenth of data record, then no. leaf pages are $0.15B$. So cost incurred is $D + D\log_F(0.15B)$.

- Range Search: It includes equality search and matching pages. So cost is $D \log_F(\text{index size}) + D * \text{matching records}$.
- Insert: Time required is for searching the page, fetching it, adding records and writing back the page. So cost is $3D + D \log_F(\text{index size})$.
- Delete: First we search for the page where record to be deleted is located, then fetch the page, remove record and write back the page. So cost is $\text{Search} + 2D$.

Unclustered Hash Index

Assumptions: the size of one data entry is 10% the size of one record; static hashing, no overflow pages (one bucket is one page); $4/5 = 80\%$ occupancy; therefore, $0.1 * 1.25B = 0.125B$ pages for data entries

- Scan: Here each record takes D time to read from a single page. So reading R record from a page takes DR time. Hence total cost for B pages is $BDR + \text{Read index}$.
- Equality Search: If search is on the search key of hashed file, then total cost is of only getting the relevant page of data entry and record, so cost is $2D$.
- Range Search: This search can be as worst as scanning the whole file. Hence cost incurred in this is of retrieving all the pages. So cost is BD .
- Insert: Here by using search key, we can read the relevant pages, add record to it and then write back the page. So cost involved with it is $4D$.
- Delete: Cost involved with it is searching for the record, reading the page, deleting the record and writing back the page. So cost is $\text{Search} + 2D$.

Please check Lecture 2 slide 21 for more information.

Answer: The main conclusion about the five file organizations is that all five have their own advantages and disadvantages. No one file organization is uniformly superior in all situations. The choice of appropriate structures for a given data set can have a significant impact upon performance. An unordered file is best if only full file scans are desired. A hash indexed file is best if the most common operation is an equality selection. A sorted file is best if range selections are desired and the data is static; a clustered B+ tree is best if range selections are important and the data is dynamic. An unclustered B+ tree index is useful for selections over small ranges, especially if we need to cluster on another search key to support some common query.

- Using these fields as the search key, we would choose a sorted file organization or a clustered B+ tree depending on whether the data is static or not.
- Heap file would be the best fit in this situation.
- Using this particular field as the search key, choosing a hash indexed file would be the best..

3. **(Exercise 8.11)** Consider the following relations:

$\text{Emp}(\text{eid: integer, ename: varchar, sal: integer, age: integer, did: integer})$

$\text{Dept}(\text{did: integer, budget: integer, floor: integer, mgr_eid: integer})$

Salaries range from \$10,000 to \$100,000, ages vary from 20 to 80, each department has about five employees on average, there are 10 floors, and budgets vary from \$10,000 to \$1 million. You can assume uniform distributions of values.

For each of the following queries, which of the listed index choices would you choose to speed up the query? If your database system does not consider index-only plans (i.e., data records are always retrieved even if enough information is available in the index entry), how would your answer change? Explain briefly.

1. Query: Print *ename*, *age*, and *sal* for all employees.

- (a) Clustered hash index on (*ename*, *age*, *sal*) fields of *Emp*.
- (b) Unclustered hash index on (*ename*, *age*, *sal*) fields of *Emp*.
- (c) Clustered B+ tree index on (*ename*, *age*, *sal*) fields of *Emp*.
- (d) Unclustered hash index on (*eid*, *did*) fields of *Emp*.
- (e) No index.

2. Query: *Find the dids of departments that are on the 10th floor and have a budget of less than \$15,000.*

- (a) Clustered hash index on the *floor* field of *Dept*.
- (b) Unclustered hash index on the *floor* field of *Dept*.
- (c) Clustered B+ tree index on (*floor*, *budget*) fields of *Dept*.
- (d) Clustered B+ tree index on the *budget* field of *Dept*.
- (e) No index.

Answer:

- 1. *We should create an unclustered hash index on _ename, age, sal_ fields of Emp (b) since then we could do an index only scan. If our system does not include index only plans then we shouldn't create an index for this query (e). Since this query requires us to access all the Emp records, an index won't help us any, and so should we access the records using a filescan.*
- 2. *We should create a clustered dense B+ tree index (c) on _floor, budget_ fields of Dept, since the records would be ordered on these fields then. So when executing this query, the first record with floor = 10 must be retrieved, and then the other records with floor = 10 can be read in order of budget. Note that this plan, which is the best for this query, is not an index-only plan (must look up dids).*