8

OVERVIEW OF STORAGE AND INDEXING

Exercise 8.1 Answer the following questions about data on external storage in a DBMS:

1. Why does a DBMS store data on external storage?
2. Why are I/O costs important in a DBMS?
3. What is a record id? Given a record's id, how many I/Os are needed to fetch it into main memory?
4. What is the role of the buffer manager in a DBMS? What is the role of the disk space manager? How do these layers interact with the file and access methods layer?

Answer 8.1 Not yet available.

Exercise 8.2 Answer the following questions about files and indexes:

1. What operations are supported by the file of records abstraction?
2. What is an index on a file of records? What is a search key for an index? Why do we need indexes?
3. What alternatives are available for the data entries in an index?
4. What is the difference between a primary index and a secondary index? What is a duplicate data entry in an index? Can a primary index contain duplicates?
5. What is the difference between a clustered index and an unclustered index? If an index contains data records as 'data entries,' can it be unclustered?
6. How many clustered indexes can you create on a file? Would you always create at least one clustered index for a file?

sid	*name*	*login*	*age*	*gpa*
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	19	3.2
53650	Smith	smith@math	19	3.8

Figure 8.1 An Instance of the Students Relation, Sorted by *age*

7. Consider Alternatives (1), (2) and (3) for 'data entries' in an index, as discussed in Section XXX. Are all of them suitable for secondary indexes? Explain.

Answer 8.2 Answer omitted.

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 8.3 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. At this time,using index is worse than file scan.

Exercise 8.4 Consider the instance of the Students relation shown in Figure 8.1, sorted by *age*: For the purposes of this question, assume that these tuples are stored in a sorted file in the order shown; the first tuple is on page 1 the second tuple is also on page 1; and so on. Each page can store up to three data records; so the fourth tuple is on page 2.

Explain what the data entries in each of the following indexes contain. If the order of entries is significant, say so and explain why. If such an index cannot be constructed, say so and explain why.

1. An unclustered index on *age* using Alternative (1).
2. An unclustered index on *age* using Alternative (2).
3. An unclustered index on *age* using Alternative (3).
4. A clustered index on *age* using Alternative (1).
5. A clustered index on *age* using Alternative (2).

6. A clustered index on *age* using Alternative (3).
7. An unclustered index on *gpa* using Alternative (1).
8. An unclustered index on *gpa* using Alternative (2).
9. An unclustered index on *gpa* using Alternative (3).
10. A clustered index on *gpa* using Alternative (1).
11. A clustered index on *gpa* using Alternative (2).
12. A clustered index on *gpa* using Alternative (3).

Answer 8.4 Answer omitted.

Exercise 8.5 Explain the difference between Hash indexes and B+-tree indexes. In particular, discuss how equality and range searches work, using an example.

Answer 8.5 Not yet available.

Exercise 8.6 Fill in the I/O costs in Figure 8.2.

File Type	*Scan*	*Equality Search*	*Range Search*	*Insert*	*Delete*
Heap file					
Sorted file					
Clustered file					
Unclustered tree index					
Unclustered hash index					

Figure 8.2 I/O Cost Comparison

Answer 8.6 Answer omitted.

Exercise 8.7 If you were about to create an index on a relation, what considerations would guide your choice? Discuss:

1. The choice of primary index.
2. Clustered versus unclustered indexes.
3. Hash versus tree indexes.

4. The use of a sorted file rather than a tree-based index.
5. Choice of search key for the index. What is a composite search key, and what considerations are made in choosing composite search keys? What are index-only plans, and what is the influence of potential index-only evaluation plans on the choice of search key for an index?

Answer 8.7 The choice of the primary key is made based on the semantics of the data. If we need to retrieve records based on the value of the primary key, as is likely, we should build an index using this as the search key. If we need to retrieve records based on the values of fields that do not constitute the primary key, we build (by definition) a secondary index using (the combination of) these fields as the search key.

A clustered index offers much better range query performance, but essentially the same equality search performance (modulo duplicates) as an unclustered index. Further, a clustered index is typically more expensive to maintain than an unclustered index. Therefore, we should make an index be clustered only if range queries are important on its search key. At most one of the indexes on a relation can be clustered, and if range queries are anticipated on more than one combination of fields, we have to choose the combination that is most important and make that be the search key of the clustered index.

Exercise 8.8 Consider a delete specified using an equality condition. For each of the five file organizations, what is the cost if no record qualifies? What is the cost if the condition is not on a key?

Answer 8.8 Answer omitted.

Exercise 8.9 What main conclusions can you draw from the discussion of the five basic file organizations discussed in Section **??**? Which of the five organizations would you choose for a file where the most frequent operations are as follows?

1. Search for records based on a range of field values.
2. Perform inserts and scans, where the order of records does not matter.
3. Search for a record based on a particular field value.

Answer 8.9 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.

1. 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.
2. Heap file would be the best fit in this situation.
3. Using this particular field as the search key, choosing a hash indexed file would be the best.

Exercise 8.10 Consider the following relation:

Emp(*eid:* `integer`, *sal:* `integer`, *age:* `real`, *did:* `integer`)

There is a clustered index on *eid* and an unclustered index on *age*.

1. How would you use the indexes to enforce the constraint that *eid* is a key?
2. Give an example of an update that is *definitely speeded up* because of the available indexes. (English description is sufficient.)
3. Give an example of an update that is *definitely slowed down* because of the indexes. (English description is sufficient.)
4. Can you give an example of an update that is neither speeded up nor slowed down by the indexes?

Answer 8.10 Answer omitted.

Exercise 8.11 Consider the following relations:

Emp(*eid:* `integer`, *ename:* `varchar`, *sal:* `integer`, *age:* `integer`, *did:* `integer`)
Dept(*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 $\langle ename, age, sal \rangle$ fields of Emp.

(b) Unclustered hash index on $\langle ename, age, sal \rangle$ fields of Emp.

(c) Clustered B+ tree index on $\langle ename, age, sal \rangle$ fields of Emp.

(d) Unclustered hash index on $\langle eid, did \rangle$ 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 $\langle floor, budget \rangle$ fields of Dept.

(d) Clustered B+ tree index on the *budget* field of Dept.

(e) No index.

Answer 8.11 The answer to each question is given below.

1. We should create an unclustered hash index on $\langle ename, age, sal \rangle$ 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 $\langle floor, budget \rangle$ 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).

3. We should create a hash index on the *eid* field of Emp (d) since then we can do a filescan on Dept and hash each manager id into Emp and check to see if the salary is greater than 12,000 dollars. None of the indexes offered lend themselves to index-only scans, so it doesn't matter if they are allowed or not.

4. For this query we should create a dense clustered B+ tree (c) index on $\langle did, sal \rangle$ fields of the Emp relation, so we can do an index only scan. (An unclustered index would be sufficient, but is not included in the list of index choices for this question.) If index-only scans are not allowed, we should then create a clustered sparse B+ tree index on the *did* field of Emp (a). This index will be just as efficient as an index on $\langle did, sal \rangle$ since both will involve accessing the data records in *did* order. However, since we now have only one attribute in the search key, it will be updated less often.