

Chapter 11: Indexing and Hashing

Database System Concepts, 6th Ed.

©Silberschatz, Korth and Sudarshan See <u>www.db-book.com</u> for conditions on re-use



Chapter 12: Indexing and Hashing

- Basic Concepts
- Ordered Indices
- B+-Tree Index Files
 - B-Tree Index Files
- Multiple-Key Access and Bitmap indices
- Hashing
 - Static Hashing
 - Dynamic Hashing
- Comparison of Ordered Indexing and Hashing
- Index Definition in SQL
- Index Definition in Oracle



Basic Concepts

Indexing mechanisms used to speed up access to desired data.

- E.g., author catalog in library
- Search Key attribute to set of attributes used to look up records in a file.
- An index file consists of records (called index entries) of the form

search-key	pointer
------------	---------

- Index files are typically much smaller than the original file
- Two basic kinds of indices:
 - Ordered indices: search keys are stored in sorted order
 - Hash indices: search keys are distributed uniformly across "buckets" using a "hash function".



Index Evaluation Metrics

- Access time
- Insertion time
- Deletion time
- Space overhead
- Access types supported efficiently. E.g.,
 - records with a specified value in the attribute
 - or records with an attribute value falling in a specified range of values.
 - This strongly influences the choice of index, and depends on usage



Ordered Indices

- In an ordered index, index entries are stored sorted on the search key value. E.g., author catalog in library.
- Primary index: in a sequentially ordered file, the index whose search key specifies the sequential order of the file.
 - Also called **clustering index**
 - The search key of a primary index is usually but not necessarily the primary key.
- Secondary index: an index whose search key specifies an order different from the sequential order of the file. Also called non-clustering index.
- Index-sequential file: ordered sequential file with a primary index.



Dense Index Files

- Dense index Index record appears for every search-key value in the file.
- E.g. index on *ID* attribute of *instructor* relation (primary index)

10101	_	 10101	Srinivasan	Comp. Sci.	65000	 \geq
12121		 12121	Wu	Finance	90000	Ś
15151		 15151	Mozart	Music	40000	~ ~
22222	_	 22222	Einstein	Physics	95000	~ ~
32343		 32343	El Said	History	60000	~ ~
33456	_	 33456	Gold	Physics	87000	~ ~
45565	-	 45565	Katz	Comp. Sci.	75000	~ ~
58583	_	 58583	Califieri	History	62000	~ ~
76543	_	 76543	Singh	Finance	80000	~ ~
76766	_	 76766	Crick	Biology	72000	~ ~
83821	_	 83821	Brandt	Comp. Sci.	92000	` ``
98345	_	 98345	Kim	Elec. Eng.	80000	



Dense Index Files (Cont.)

Dense index on dept_name, with instructor file sorted on dept_name (primary index)

Biology	_	├ →	76766	Crick	Biology	72000	
Comp. Sci.	-		10101	Srinivasan	Comp. Sci.	65000	
Elec. Eng.			45565	Katz	Comp. Sci.	75000	<u>×</u>
Finance			83821	Brandt	Comp. Sci.	92000	×
History	\backslash		98345	Kim	Elec. Eng.	80000	×
Music			12121	Wu	Finance	90000	×
Physics	$\left \right\rangle$		76543	Singh	Finance	80000	×
	/		32343	El Said	History	60000	×
			58583	Califieri	History	62000	×
			15151	Mozart	Music	40000	×
		\searrow	22222	Einstein	Physics	95000	×
			33465	Gold	Physics	87000	*



Sparse Index Files

Sparse Index: contains index records for only some search-key values.

- Only applicable when records are sequentially ordered on search-key (i.e. in primary index)
- To locate a record with search-key value *K* we:
 - Find index record with largest search-key value < K
 - Search file sequentially starting at the record to which the index record points

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	
	10101 12121 15151 22222 32343 33456 45565 58583 76543 76543 76766 83821 98345	10101 Srinivasan 12121 Wu 15151 Mozart 22222 Einstein 32343 El Said 33456 Gold 45565 Katz 58583 Califieri 76543 Singh 76766 Crick 83821 Brandt 98345 Kim	10101SrinivasanComp. Sci.12121WuFinance15151MozartMusic22222EinsteinPhysics32343El SaidHistory33456GoldPhysics45565KatzComp. Sci.58583CalifieriHistory76543SinghFinance76766CrickBiology83821BrandtComp. Sci.98345KimElec. Eng.	10101 Srinivasan Comp. Sci. 65000 12121 Wu Finance 90000 15151 Mozart Music 40000 22222 Einstein Physics 95000 32343 El Said History 60000 33456 Gold Physics 87000 45565 Katz Comp. Sci. 75000 58583 Califieri History 62000 76543 Singh Finance 80000 76766 Crick Biology 72000 83821 Brandt Comp. Sci. 92000 98345 Kim Elec. Eng. 80000

Database System Concepts - 6th Edition



Sparse Index Files (Cont.)

- Compared to dense indices:
 - Less space and less maintenance overhead for insertions and deletions.
 - Generally slower than dense index for locating records.
- Good tradeoff: sparse index with an index entry for every block in file, corresponding to least search-key value in the block.





Secondary Indices Example



Secondary index on *salary* field of *instructor*

- Index record points to a bucket that contains pointers to all the actual records with that particular search-key value.
- Secondary indices have to be dense



Primary and Secondary Indices

- Indices offer substantial benefits when searching for records.
- BUT: Updating indices imposes overhead on database modification --when a file is modified, every index on the file must be updated,
- Sequential scan using primary index is efficient, but a sequential scan using a secondary index is expensive
 - Each record access may fetch a new block from disk
 - Block fetch requires about 5 to 10 milliseconds, versus about 100 nanoseconds for memory access



Multilevel Index

- If primary index does not fit in memory, access becomes expensive.
- Solution: treat primary index kept on disk as a sequential file and construct a sparse index on it.
 - outer index a sparse index of primary index
 - inner index the primary index file
- If even outer index is too large to fit in main memory, yet another level of index can be created, and so on.
- Indices at all levels must be updated on insertion or deletion from the file.



Multilevel Index (Cont.)





Index Update: Deletion

10101 10101 Srinivasan Comp. Sci. 65000 32343 12121 Wu Finance 90000 76766 15151 Mozart Music 40000 22222 Einstein 95000 Physics El Said History 32343 60000 If deleted record was the **Physics** 33456 Gold 87000 only record in the file with its Comp. Sci. 75000 45565 Katz particular search-key value, 58583 Califieri History 62000 76543 Singh Finance 80000 the search-key is deleted Crick 76766 Biology 72000 from the index also. Brandt Comp. Sci. 83821 92000 98345 Kim Elec. Eng. 80000

Single-level index entry deletion:

Dense indices – deletion of search-key is similar to file record deletion.

Sparse indices –

- if an entry for the search key exists in the index, it is deleted by replacing the entry in the index with the next search-key value in the file (in search-key order).
- If the next search-key value already has an index entry, the entry is deleted instead of being replaced.

Database System Concepts - 6th Edition



Index Update: Insertion

Single-level index insertion:

- Perform a lookup using the search-key value appearing in the record to be inserted.
- Dense indices if the search-key value does not appear in the index, insert it.
- Sparse indices if index stores an entry for each block of the file, no change needs to be made to the index unless a new block is created.
 - If a new block is created, the first search-key value appearing in the new block is inserted into the index.
- **Multilevel insertion and deletion:** algorithms are simple extensions of the single-level algorithms



Secondary Indices

Frequently, one wants to find all the records whose values in a certain field (which is not the search-key of the primary index) satisfy some condition.

- Example 1: In the *instructor* relation stored sequentially by ID, we may want to find all instructors in a particular department
- Example 2: as above, but where we want to find all instructors with a specified salary or with salary in a specified range of values
- We can have a secondary index with an index record for each search-key value