Indices

File Organization and Indexing

Assume that we have a large amount of data in our database which lives on a hard drive(s)

What are some of the things we might wish to do with the data?

- Scan: Fetch all records from disk
- Equality search
- Range search
- Insert a record
- Delete a record

How expensive are these operations? (in terms of execution time)

How expensive are these operations?

The cost of operations listed below depends on how we organize data.

There are three main ways we could organize the data

- Heap Files
- Sorted File (Tree Based Indexing)
- Hash Based Indexing

Scan/ Equality search/ Range selection/ Insert a record/ Delete a record

Important Point

Data which is organized based on one field, may be difficult to search based on a different field.

Consider a phone book. The data is well organized if you want to find Eamonn Keogh's phone number, suppose to want to find out whose number 234-2342 is?

Informally, the attribute we are most interested in searching is called the **search key**, or just **key** (we will formalize this notation later).

Note that the search key can be a combination of fields, for example phone books are organized by <*Last_name*, *First_name*>

Unfortunately, the word **key** is overloaded in databases, the word **key** in this context, has nothing to do with primary key, candidate key etc.

Basic Concepts

- Indexing mechanisms used to speed up access to desired data.
  - E.g., author catalog in library
- **Search Key -** attribute (or set of attributes) used to look up records in a file.
- An **index file** consists of records (called **index entries, or data 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 (I.e tree based)
  - **Hash indices:** search keys are distributed uniformly across "buckets" using a "hash function".

Ordered Indices

- In an **ordered index,** (I.e tree based) 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.
  - Note that we can have at most one primary index
- **Secondary index**: an index whose search key specifies an order different from the sequential order of the file. Also called non-clustering index**.**
  - We can have as many secondary indices as we like.
  - Many times secondary index are not updated after deletion

Operations in primary indices

- RAM
- DISK

- Create Index
  - Read data file and create array with Keys
  - Sort array
  - Save array (at some point – bulk insert)
- Load Index
  - Load Index (sequential read) → Index in RAM
- Search for Register (key)
  - Binary search in index. Get file offset
  - Access record (seek + read)

Operations in primary indices II

- RAM
- DISK

- All index records have the same length
- Insert Record
  - Data File: add record and update pointers
  - Index: add key keeping order
- Delete Record
  - Data File: “delete” record (update pointers)
  - Index: delete key. Compact index
- Update Field
  - Data File: update field
  - Index: if part of key (delete and insert). Even better: do NOT update PK
  - Index: data update may modify record address. Update if needed

Operations in secondary indices

- RAM
- DISK

- All index records have the same length
- Search Record
  - Search key in secondary index. Get primary key
  - Search key in primary index. Get data offset
  - Access record (seek + read)
- Insert Record
  - As primary key insert
  - Same strategy than primary key insert
- Delete Field
  - As primary key delete
  - Index: delay deletion
  - Deleted register will be detected when access to PK
  - Periodically clean secondary indices

Search Cost

- $\sigma_{(A=a)}$ (search key) → O(logn)
- $\sigma_{(A>a \text{ and } A< b)}$, $\sigma_{(A=a \text{ and } B= b)}$ (range search) → O(n)
- $A \cap B$ → O(n)
- A NJ B → O(n)
- A x B → $O(n^2)$

Exercise: Employee DataBase

Offset (bytes)	SSN	Name	Family Name	Age	Gender	Dept
100	45	Arturo	Rodríguez	33	M	I
129	65	Elena	Pérez	23	F	I
152	25	Carlos	Gutiérrez	43	M	HR
181	55	Beatriz	Pérez	54	F	C
206	75	Ana	Gómez	23	F	C
227	35	Luis	Gómez	34	M	M
249	15	Carlos	Pérez	19	M	M

- Create a primary index plus two secondary indexes (fields Gender and Dept)
- Describe how to perform the query: *Female employees that work in department "I"* using the indexes
- Which indexes need to be updated if Luis is deleted. Assume we want to minimize computational cost
- Which indexes need to be updated if Pepe is added. Assume we want to minimize computational cost
- Update a register (PK, Index, other)
- Cost of search in this indexes.

PK	OFFSET
15	249
25	152
35	227
45	100
55	181
65	129
75	206

GENDER	PK
M	15
M	25
M	35
M	45
F	55
F	65
F	75

DEPT	PK
C	55
C	75
I	45
I	65
M	15
M	35
HR	25

- Describe how to perform the query: *Female employees that work in department "I"* using the indexes
    - PK NJ $\sigma_{(gender=F)}$ GenderIndex ∩ $\sigma_{(dept=I)}$ DeptIndex
- Which indexes need to be updated if Luis is deleted. Assume we want to minimize computational cost
    - PKIndex only
- Cost of search in this indexes.
    - Log (binary search)

Indices are great (if they fit in memory)

If not… block (sparse) or multilevel indices

B-Trees