Indexes

CISC637 Lecture #13

Ben Carterette

Recap

- Database algorithm computation model:
  – All records in a relation are stored in a file on disk
  – DBMS asks the operating system for *x* bytes off disk
  – The OS reads data off disk in *blocks* of 4K into RAM
  – Then the *x* bytes the DBMS asked for are provided from RAM

- Database algorithm analysis model:
  – The goal is to minimize the number of block reads
  – Variables:
    - **D** represents time it takes to read one block off disk
    - **B** is the number of blocks that a file is stored in
    - **C** is the time it takes to process one record in the CPU
    - **R** is the average number of records in a block
  – Total time to scan a full file = D*B + C*(R*B)
  – Since D, C are treated as constants, and D $\gg$ C, we analyze algorithms in terms of O(B)

Recap

- Six operations to analyze:
  - Full file scan
    `SELECT * FROM Student`
  - Equality search on key
    `SELECT * FROM Student WHERE ID = 10101`
  - Range search on key
    `SELECT * FROM Student WHERE ID > 20000 AND ID < 40000`
  - Insert
    `INSERT INTO Student VALUES (…)`
  - Update non-key field
    `UPDATE Student SET name="Bob" WHERE ID=10101`
  - Delete
    `DELETE FROM Student WHERE ID=10101`

Recap

- Two ways to store records in a file:
  - Heap file: records stored in random order
  - Sequential file: records stored in sorted order (usually sorted by primary key values)

	heap	sequential
scan	O(B)	O(B)
= search	O(B)	O($\log_2$ B)
<> search	O(B)	O($\log_2$ B + m)
insert	O(1)	O($\log_2$ B)
update	O(B)	O($\log_2$ B)
delete	O(B)	O($\log_2$ B)

$\log_2$ B is possible using the binary search algorithm

Indexes

- A separate data structure that speeds up access to data
  - Each index record has three pieces of data:
    (search key, pointer(s) to block(s) where record is stored, offset(s) to record within block)
    Note that search key does not need to be a primary key
- Like tables, index data is stored in files on disk
  - An index is a special kind of table
- Index files are typically much smaller than original table files
  - Assume index record = 10% size of actual record
  - Index file size ≤ 10% of table file size
- Two types:
  - **Ordered index** has search keys in sorted order (like a sequential file)
  - **Hash index** places search keys in buckets according to a hash function

Primary and Secondary Indexes

- A **primary index** is an ordered index in which index records are stored in the same order as records in a sequential file
  - Also called a **clustering index**
  - *Not necessarily* an index on the primary key
    - In MySQL, all tables have their primary index on the primary key
    - That is not the case in all DBMS
- A **secondary index** is any other index
  - Also called **non-clustering index**

Example 1: Primary Index on Primary Key

```
SELECT * FROM Instructor WHERE ID=76766
```

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	block 1
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	block 2
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	block 3
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

Example 2: Primary Index on non-Primary Key

- Primary index on dept_name
  - Primary index **does *not* have to be** on primary key
  - Primary index **does** have to be on the field(s) the file is sorted on

Biology
Comp. Sci.
Elec. Eng.
Finance
History
Music
Physics

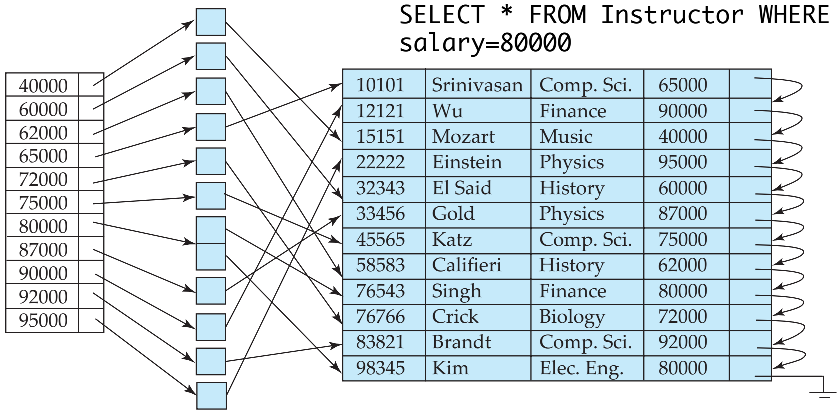
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
33465	Gold	Physics	87000

```
SELECT * FROM Instructor WHERE dept_name=Comp.Sci.
```

Example 3: Secondary Index

- Secondary index on salary
  - A secondary index is on a non-sequential field

```
SELECT * FROM Instructor WHERE
salary=80000
```



Creating Indexes in MySQL

```
CREATE INDEX indexName
  ON Table (field [, …])
  USING [HASH|BTREE]
```

- Creates a B+-tree index on the specified column(s)
  - HASH only works for tables defined with `Engine=MEMORY`
- MySQL automatically maintains the index along with inserts/updates/deletes on the table
- MySQL creates primary indexes on primary keys by default
  - If no primary key, primary index on first UNIQUE NOT NULL field
  - If no UNIQUE NOT NULL field, primary index on internal row ID
- Secondary indexes on foreign key fields are also created by default

Algorithm Analysis Model

- Recall:
  - B is the number of blocks
  - D is the average time to read/write a block from/to disk
  - R is the number of records per block
  - C is average time to process a record at the CPU
- Also:
  - B_I is the number of blocks required for an index
    - Usually assume $B_I = B/10$ or $B/8$
  - R_I is the number of index records per index block
    - Usually assume $R_I = 10R$ or $8R$

 17

Heap File

- Records stored in random order, no index
  - File stored across B blocks with R records per block
  - (For insert, assume we can locate a block with free space in constant O(1) time)
- Time costs:
  - Scan?
  - Search with equality?
  - Search for range?
  - Insert?
  - Delete?

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

Heap file on any search key except ID

Heap File

- Records stored in random order, no index
  - (For insert, assume we can locate a block with free space in O(1) time)
- Time costs:
  - Scan? DB + BRC
    - (DB = total time to read all B pages plus BRC = total time to process all R records in each page)
  - Search with equality? DB + BRC
    - (full scan necessary)
  - Search for range? DB + BRC
    - (full scan necessary)
  - Insert? 2D + C
    - (go to free block, read it, add record, write it)
  - Delete? (DB + BRC) + C + D
    - (first search for record to be deleted, delete it, write block back to disk)

Sequential File With No Index

- Records stored in order of a search key, no index
  - (Assume records sorted contiguously in blocks—no jumping between blocks)
  - (Also assume that we can specify block to start an operation at, e.g. we could start at 5^{th} block of file)
- Time costs:
  - Scan?
  - Search with equality?
  - Search for range?
  - Insert?
  - Delete?

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

Sequential file on ID

Sequential File With No Index

- Records stored in order of a search key, no index
  - (Assume records sorted contiguously in blocks—no jumping between blocks)
  - (Also assume that we can specify block to start an operation at)

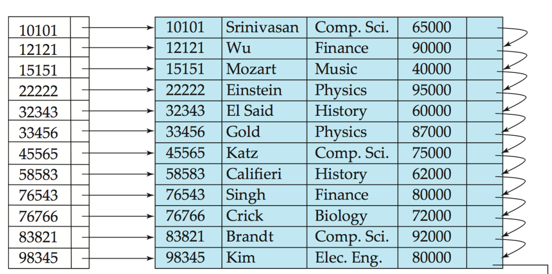
- Time costs:
  - Scan? DB + BRC
  - Search with equality? (D+2C) log B + C log R
    - (use binary search with first+last record in blocks to find the right block, then use binary search within records in that block to find the record)
  - Search for range? (D+2C) log B + B_m(D+RC)
    - B_m = total number of blocks to store all records in range; assume $B_m \ll B$
    - (same as search with equality, except that we may have to read more blocks to complete result)
  - Insert? in worst case, scan entire file and write it back to disk
    - (to maintain ordering of records for binary search)
  - Delete? equality search + D + C
    - (find block containing record to delete, delete record, write block to disk)

Sequential File With Primary Index

- Assume search key is primary key, so there is one index record per actual record

- Assume index records are 10% size of actual records

- Space costs:
  - Overhead?
- Time costs:
  - Scan?
  - Search with equality?
  - Search for range?
  - Insert?
  - Delete?



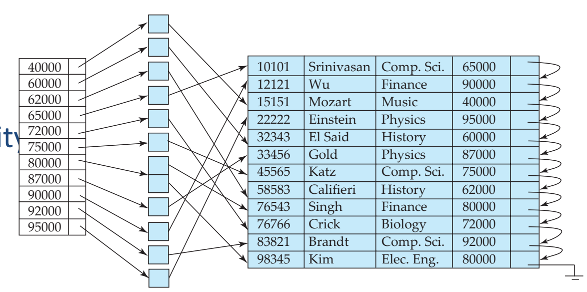
Sequential file with primary index on ID

Sequential File With Primary Index

- Assume search key is primary key, so there is one index record per actual record
- Assume index records are 10% size of actual records
- Space costs:
  - Overhead? 0.1B if index blocks 100% full, 0.125B if 80% full
- Time costs:
  - Scan? B(D+RC) for full table scan, $B_I(D+R_IC)$ for index scan
    - Index scan is useful if you only need the indexed fields
  - Search with equality? $(D+2C) \log B_I + C \log R_I + (D + C)$
  - Search for range? $(D+2C) \log B_I + C \log R_I + B_m(D + RC)$
  - Insert? $2B_I(D+R_IC)$ + time to insert in main file
    - read and re-write entire index to keep it in sorted order
    - write new record to free space, update pointers in sequential file
      - (requiring at most two additional disk reads and writes)
  - Delete? find in index and delete, then delete in file

Secondary Index

- Assume search key is not primary key, so there may be more than one actual record per index record
- Space costs:
  - Overhead?
- Time costs:
  - Scan?
  - Search with equality?
  - Search for range?
  - Insert?
  - Delete?



Sequential file with secondary index on salary

Secondary Index

- Assume search key is not primary key, so there may be more than one actual record per index record
- Space costs:
  - Overhead? depends on number of unique values in search key fields
- Time costs:
  - Scan? $B_I(D+R_IC)$ for index scan
    - an index scan of a secondary index will give you the primary key values as well as the values of the search key
  - Search with equality? $(D+2C) \log B_I + C \log R_I + B_m(D + RC)$
  - Search for range? $(D+2C) \log B_I + C \log R_I + B_m(D + RC)$
  - Insert? $2B_I(D+R_IC)$ + time to insert in main file
    - read and re-write entire index to keep it in sorted order
  - Delete? find in index and delete, then delete in file

Analysis Table

	heap	sequential	seq + primary index	secondary index
scan	O(B)	O(B)	$O(B_I)$ index scan	$O(B_I)$ index scan
= search	O(B)	$O(\log_2 B)$	$O(\log_2 B_I)$	$O(\log_2 B_I + m)$
<> search	O(B)	$O(\log_2 B + m)$	$O(\log_2 B_I + m)$	$O(\log_2 B_I + m)$
insert	O(1)	$O(\log_2 B)$	$O(\log_2 B_I)$	$O(\log_2 B_I)$
update	O(B)	$O(\log_2 B)$	$O(\log_2 B_I)$	$O(\log_2 B_I)$
delete	O(B)	$O(\log_2 B)$	$O(\log_2 B_I)$	$O(\log_2 B_I)$

Notes:
- Sequential files need to be re-sorted after inserts—$\log_2 B$ is the best case assuming the file is sorted, worst case is O(B)
- Indexes also need to be re-sorted after inserts—if they are not, $O(B_I)$ is the worst-case time for searching
  - We can do better… in more ways than one