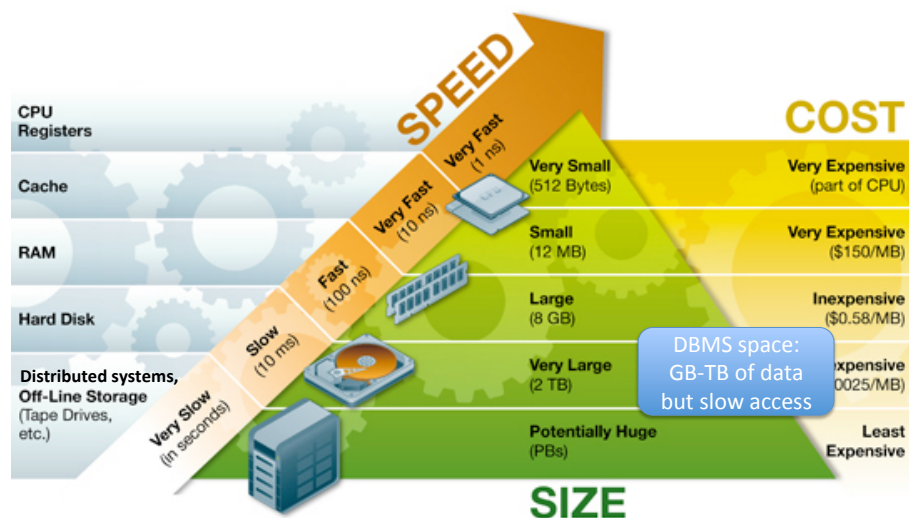


File Organization and Storage Access

CISC637, Lecture #12

Ben Carterette

Computer Storage Hierarchy



Disk Access

`SELECT * FROM student WHERE name='Lee'`

- Assume student records are stored in a file on disk
- For DBMS to perform this SELECT, it must:
 - Initialize result set
 - Open student file
 - Loop until EOF:
 - Read record from file
 - Check if record.name == 'Lee'
 - If so, add to result set
 - Close file
 - Return result set

File Storage and Disk Access

- Suppose we define student table like this:

```
CREATE TABLE student (
  ID INT,
  name CHAR(100),
  dept_name CHAR(100),
  tot_cred INT,
  PRIMARY KEY (ID),
  FOREIGN KEY (dept_name) REFERENCES department (dept_name))
```

- Record size = 4+100+100+4+a little more for metadata
 - Let's say record size = 256 bytes
- File size:
 - 100 students: 25K
 - 1000 students: 250K
 - 10,000 students: 2.4M
 - 100,000 students: 24M
 - ...

File Storage and Disk Access

`SELECT * FROM student WHERE name='Lee'`

- DBMS asks operating system to retrieve one record (256 bytes) at a time
- Operating systems will not read 256 bytes off disk at a time
 - OSes read data off disk in *pages* or *blocks*—typically 4K at a time
 - For this example, each 4K page would contain 16 separate student records
- File storage:
 - 100 students stored in 7 pages
 - 10,000 students stored in 625 pages
 - 100,000 students in 6250 pages
- To complete the SELECT, the OS needs to read all pages off disk
 - For a typical HDD, 6250 pages * 10ms/page = 62500ms = 62.5s
 - For a typical SSD, 6250 pages * 1ms/page = 6250ms = 6.5s
 - Either way, not very fast
 - For comparison, time to check whether each record has name == 'Lee' is trivial

Minimize Disk Access

- Disk access is time-consuming
- Things the DBMS can do to help minimize it:
 - Keep data in main memory when possible
 - Organize records in files in ways that reduce disk access time
 - Use additional data to help make disk access of primary data more direct

Buffer Manager

- The **buffer** is a part of main memory used for temporary storage of data on disk
- The **buffer manager** is a subsystem that controls disk access
 - When DBMS needs to access data on disk, it calls the buffer manager
 - If the block containing that data is in the buffer, the buffer manager returns the memory address
 - Otherwise,
 - manager allocates space in buffer
 - reads block into buffer and returns address

Buffer Manager

- If there is no space in buffer, the buffer manager must free some
 - Replace blocks currently in buffer
 - If blocks have changed since last read, write changes to disk
 - “Page fault”: attempt to access a block that has been replaced in the buffer

File Organization

- File is stored in blocks on disk, each block contains records organized in some way
- Three approaches:
 - “heap” – random organization of records in block
 - sequential – keeps records in block in sorted order by some key value
 - hash – result of a hash function computed on some attribute gives location of record in block

Heap File Organization


- Records stored in the order added

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 Organization

- Add extra data to each record pointing to location of next record in sequence

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	

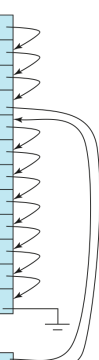


Sequential File Organization

- To delete, alter pointer sequence
- To insert, locate position for new record
 - if space, write and alter pointers
 - if not, add data to overflow block

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	

32222	Verdi	Music	48000	
-------	-------	-------	-------	--



MySQL File Storage

- Every table has a corresponding .frm file on disk, consisting of:
 - Header containing basic information about the table
 - (MySQL version, table length, character set, time created, average record length, etc)
 - Information about keys
 - Comments, if any given by table creator
 - Column information (including name, data type/ domain, and more information)

MySQL File Storage

- Actual record storage depends on engine and configuration options
 - MyISAM creates two files for each table (table.MYD, table.MYI); .MYD file stores records
 - InnoDB can either store all records in one big file (ibdata1), or have a separate file for each table (table.ibd)
- I will cover InnoDB with separate files

InnoDB table.ibd File

- File is stored across multiple blocks (*pages* in MySQL documentation)
 - Each block has a fixed size of 16KB
 - Each block contains:
 - a “fil header” containing pointers to the previous and next block in the file
 - a “page header” including the number of records stored in the block, pointers to free space in the block for new records, and information about recent updates to data in the page
 - actual record values, stored in either a heap or in sequential order
 - a “page directory” containing pointers to some of the records in the block