COSC2406/2407: Database Systems

Disks and Files

Xiangmin (Emily) Zhou

RMIT University
Room : 14.11.04
Online Consultation via Collaborate Ultra(no appointment required): 10:20-11.20am
Thursdays
Email : xiangmin.zhou@rmit.edu.au

Lecture 2

References: Ramakrishnan & Gehrke Chapter 9
Garcia-Molina et al. Chapter 11
Elmasri & Navathe Chapters 5 & 6

Overview

Over the next two lectures, we will lay the foundations for studying database systems. We will discuss:

- Disks—how they are structured, their performance characteristics, and how they are used

- Operating systems—how they interact with disks, including the principles of interaction that are used by database management systems (DBMSs)
- Files—how files are allocated and managed by OSes and DBMSs
- Data and records—how data and records are managed in files

Representing Data Elements

First we discuss disks, their characteristics, and how DBMS buffer managers interact with disks.

We focus on blocks, and next week we focus on how data is stored in those blocks:

1. Attributes are stored as variable- or fixed-length sequences of bytes (known as *fields*)

2. Fields are stored together to form logical *records*
3. Records are stored in disk blocks
4. Blocks of records of the same type are typically stored together to form a *file* (note that a file in a DBMS is different in concept to an operating system file)

Memory Hierarchy

Cache is the lowest level of the hierarchy. Two components form the cache: on-board cache, on the same chip as the CPU, and level-2 cache on another chip. The typical maximum cache size is around one megabyte. A cache can be accessed in a few nanoseconds.

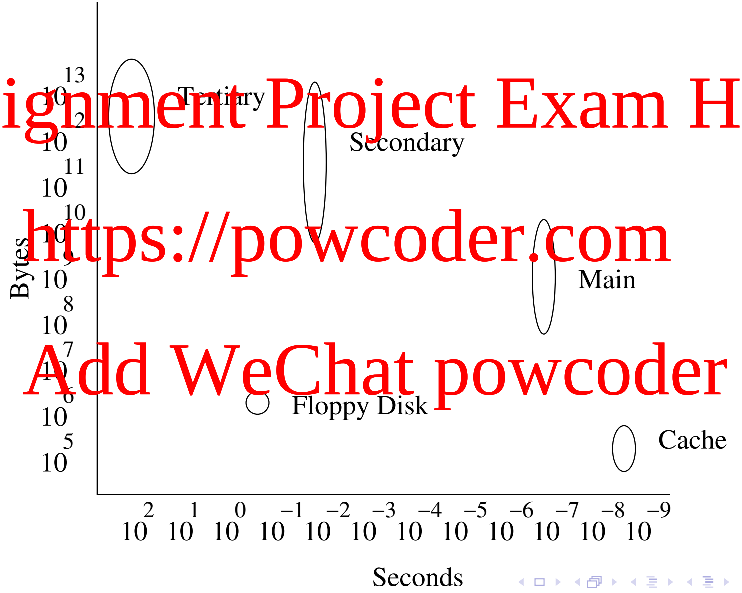
Main Memories are next in the hierarchy. A typical capacity is a few hundred megabytes, and access takes around 10–100 nanoseconds. Main memories are random access.

Secondary storage either SSD (faster) or HDD (slower but cheaper per gigabyte). Secondary memory is typically non-volatile, and also supports random access. Access takes around 10–30 milliseconds, and typical capacities are in tens of gigabytes.

Tertiary storage is cheaper still and slower again; examples are tapes—DLT, DAT, and so on—that are capable of very large storage capacities (perhaps terabytes). Access times are perhaps seconds or minutes.

Data is typically stored on disks and brought to main memory for processing by the Database Management System (DBMS).

Memory Hierarchy



Secondary Storage - HDD versus SSD

- HDD = Hard Disk Drive
- SSD = Solid State Drive
- Evolution of Secondary storage devices Hard disk, Solid State Devices, Light Based Memory Chip.
  Gopinath Vijayarangan (16 Nov 2015)
  `https://www.youtube.com/watch?v=99bS7UORXfg` (4:34)

- HDD vs SSD - What is the difference?
  Carey Holzman (9 Feb 2015)
  `https://www.youtube.com/watch?v=O4ykrNhI5xk` (11:06)

HDD: Characteristics

Disk storage *was* the most common non-volatile storage media for large amounts of data.

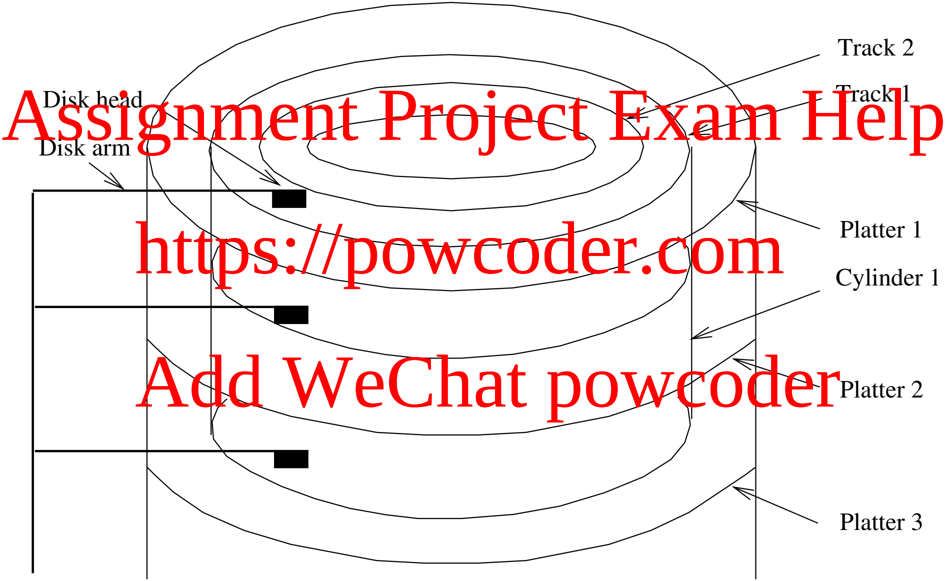
A single disk surface is divided into *tracks*, with each track containing as many as 500 *sectors*. There are as many as 20,000 tracks per surface.

Multi-platter disks group tracks one above another into *cylinders*.

The time to find a track and set-up to read or write is known as the *seek time*, whilst the spin-time to find data is called *rotational latency* (more in a moment).

Latency exists between tracks in a cylinder, as they are never perfectly aligned.

HDD: Sectors, Tracks, Platters and Cylinders



HDD Disks

Data is stored in physical *sectors*.

A physical sector is a fixed length unit of storage that can be addressed, read, or written.

The sector size is a physical characteristic of the disk and a typical sector stores thousands of bytes of data.

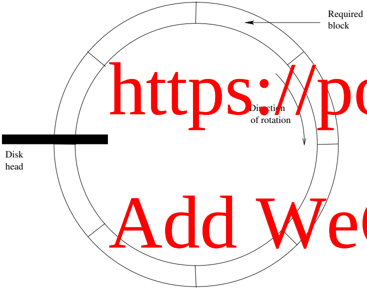
Logical *blocks* of data are stored in one or more sectors. The block size is usually set when the disk is formatted or initialised.

(See Section 9.1.1 of Ramakrishnan & Gehrke for more information on disks.)

HDD Performance

The access time for a block on disk has 3 main components:

- seek time
- rotational delay
- transfer time



For typical HDD, the *block access time* is 10 to 15 milliseconds. Since seek time and rotational delays dominate the total, the time to read one block of data is almost the same as that of reading several *contiguous* blocks. This is sometimes referred to as *blocked access*.

HDD Performance

Consider a simple disk that has a 10 ms seek time, 8 kb blocks, and can read 10 Mb per second from disk.

A: How long will it take to read a single block?

B: How long will it take to read ten contiguous blocks?

C: How long will it take to read ten non-contiguous blocks?

What does this suggest about record organisation?

HDD Performance

The simple disk has a 10 ms seek time, 8 kb blocks, and can read 10 Mb per second from disk.

A: Reading a single block: One seek, one read. Therefore,
$0.01 + \frac{8\times1024}{10\times1024^2} = 0.01 + 0.00078\text{ s} = 10.78\text{ ms}$

Ten contiguous blocks: One seek, ten reads: Therefore,
$0.01 + 10 \times 0.78 = 17.8\text{ ms}$

Ten non-contiguous blocks: Ten seeks, ten reads: Therefore,
$10 \times 0.1 + 10 \times 0.78 = 107.8\text{ ms}$

HDD Performance

Consider a disk that rotates at 7,200 rpm (it makes a rotation in $60/7200 = 0.00833$ seconds or 8.33 ms). The block size is 16,384 bytes, and disk also has 16,384 tracks per surface. There are 128 sectors per tracks, and each sector stores 4,096 bytes.

The head can start (or stop) moving in 1 ms, plus an additional 1 ms for every 1,000 cylinders travelled. The head can therefore move one track in 2.001 ms, or from the innermost track to the outermost track in $2 + 16384/1000 = 18.384$ ms.

What are the approximate minimum and maximum times to read a block?

HDD Performance

Minimum time: the time to read when the head is positioned to read the required block. So, 4 sectors of the 128 sectors require $4/128 \times 8.33 = 0.26$ ms.

Maximum time: the time to move the head as far as possible, and the block is as far away as possible when the head arrives. So, 18.384 ms to move the head. Then, 8.33 ms of latency (a full rotation, since we've just missed the start of the desired block). Finally, $4/128 \times 8.33 = 0.26$ ms to read. Total: $18.38 + 8.33 + 0.26 = 26.97$ ms.

The average time is harder: see Example 11.5 in the text.)

Improving HDD performance

The Elevator Algorithm

In practice, disks have a queue of requests for blocks.

One approach is to process them in order.

Better approach is the *elevator algorithm*, so-called because it schedules block accesses as the disk arm sweeps back and forth across the disk surface.

- When the head passes a requested track, it stops, services the request and continues its sweep.
- When there are no requests in the current direction, the disk arm sweeps in the opposite direction.

Organising Data by Cylinders

Since seek time represents about half average time to access a block, it makes sense to store data that is likely to be accessed together (such as a relation in a database), in a single cylinder.

If there is not enough room, then use several adjacent cylinders.

If reading whole cylinder, only need one seek (to move to the cylinder) and first rotational latency (until first block moves under the head).

Improving HDD performance: Striping Disks

Striping creates a single logical volume from two or more disks. As an example, when tracks are striped on two disks, odd numbered tracks come from one disk and even numbered tracks from the other.

The principle behind striping is to even out the load on disks, giving improved throughput. It also increases performance on sequential accesses, since seeking and reading can be overlapped between disks. However, this can affect other disk activities.

Striping can be in units of tracks, blocks, cylinders, sector groups, and so on (see section 9.2.1).

(Striping without redundant storage is so-called RAID-0. We discuss RAID on the next slide.)

Improving redundancy and performance: Disk Arrays

RAIDs (*Redundant Arrays of Inexpensive Disks*) are a set of disk drives that are accessed concurrently to increase transfer rates and the number of possible concurrent accesses.

RAID also offers other features in so-called RAID-1 through to RAID-6. Different RAID levels offer different levels of redundancy and performance under different load types. For example, under RAID-1, two identical copies of data are maintained (mirroring). RAID-6 uses Solomon-Reed codes to recover from up to two simultaneous failures (you should read and understand section 9.2.3).

RAID provides both redundancy and performance for HDD.

RAID also can be and is used to provide redundancy for SSD, but as not required for performance with SSD the technology is evolving.

RAID-3

For a set of n disks operating under RAID-3, one disk is set aside to store only the parity of the other $n - 1$ disks. For example, consider the first bit on each data disk. Then suppose that this data bit is set to 1 on i of the disks. We then set the corresponding bit on the check disk to 1 if i is odd, or to 0 if i is even. Therefore, the modulo-2 sum of a corresponding bit across all disks will always be zero.

If any disk fails, the parity disk and other disks can be used to reconstruct the failed disk: the bit in any position is the modulo-2 sum of the the bits in the corresponding position of the $n - 1$ other disks.

Improving HDD performance: Disk Caches

Disks (and file systems) are typically designed to reduce the number of disk accesses required to retrieve data; memory accesses are much cheaper than disk accesses.

A *disk cache* is a fast storage device that keeps data in memory from the last *n* disk accesses.

All I/O transfers in the operating system will occur between the disk cache and main memory (if a cache is present).

Generally, the *cache replacement policy* for a full cache is a variation on Least Recently Used (LRU): every reference to a block in the cache moves that block to the end of the “replacement queue”.

A block read will succeed without a disk access if the block is in the cache; just needs a simple memory copy from cache to user space.

Disk caches are capable of detecting whether recent accesses were sequential; if so, pre-fetch data blocks in anticipation.

Because of the delayed writing and anticipated reading scheme, such cache algorithms are known as *read-ahead, write-behind* caching.

HDD Performance Improvement

In summary, several techniques are used to improve disk access performance:

- Organising data by cylinders
- Using several disks instead of one
- Striping and mirroring disks
- Using the elevator algorithm
- Pre-fetching data into a cache

Disk Performance in Practice

Disk characteristics in a real system.

```
iostat -x
                                    extended disk statistics
disk         r/s   w/s    Kr/s    Kw/s wait actv   svc_t  %w   %b
sd0          1.0   1.0     9.0    11.8  0.0  0.0    23.9   0    2
sd2          1.0   0.5     7.2     3.7  0.0  0.1    47.2   0    1
sd3          1.5   0.7    10.6     5.4  0.0  0.0    22.6   0    2
sd6          0.0   0.0     0.0     0.0  0.0  0.0   271.6   0    0
sd17         0.1   0.0     0.4     0.2  0.0  0.0    42.0   0    0
```

`svc_t` is the interesting column: it indicates how long, on average, the disk takes to respond to a request in milliseconds.

Disk Space Management in a DBMS

The lowest level of the DBMS software—the *disk space manager*—hides details of the disk hardware from higher levels of the DBMS software, enabling them to regard data on a disk as a collection of *pages* (see section 9.3 of the text).

Higher levels call on this level to allocate or deallocate pages, and to read or write pages.

The page size is set to be the same as the disk block size, so each page is stored in a disk block, and a page read/write corresponds to one disk I/O.

The terms “page” and “block” are sometimes used interchangeably.

Buffer Management in a DBMS

The *buffer manager*, a level above the disk space manager, ensures that pages requested by higher levels are present in main memory (see section 9.4 of the text).

The buffer manager partitions available main memory into *frames*. One page can fit in each frame. The collection of frames is called the *buffer pool*.

For each frame in the DBMS's buffer pool, the buffer manager maintains a *pin-count* of the number of users using each disk page or *frame*.

If a new process starts using a frame, its pin-count is increased (called *pinning*). When a process is finished with the frame, the pin-count is decremented. A page can only be swapped out of memory when the pin-count is zero.

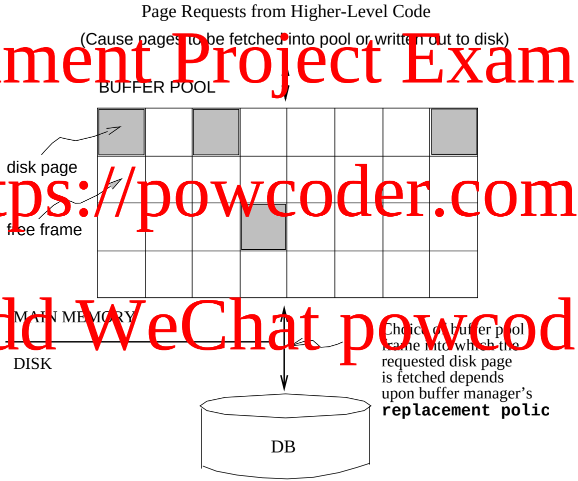
Buffer Management...

If the requestor has modified a page, it sets a "dirty" bit for that frame. Before the buffer manager can swap a page with a dirty bit out of memory, it must first be written to disk.

A variety of replacement policies are used for deciding which page should be swapped out of memory, including Least Recently Used (LRU), Most Recently Used (MRU), First-in-first-out (FIFO), Random and Clock.

Different replacement policies are optimal in different situations. (Read section 9.4.1 about the replacement algorithms.)

Buffer Management in a DBMS



Page Requests from Higher-Level Code

(Cause pages to be fetched into pool or written out to disk)

BUFFER POOL

disk page

free frame

MAIN MEMORY

DISK

DB

Choice of buffer pool frame into which the requested disk page is fetched depends upon buffer manager's **`replacement polic`**

DBMS Buffer Management in Practice

- Named pools (Sybase/DB2)—Pages can be grouped in pools, and different replacement policies assigned to the pools
- Different replacement policies (DB2)—Allows LRU, clock variants, “hated pages”, MRU for utility operations, and FIFO
- Prefetch configuration—In DB2 the number of pages pre-fetched can be configured and can be changed for different operations. Oracle 8 uses prefetch for sequential scans (much like the OS does), retrieving large objects, and certain *index* scans (more on indexes later)

Structured File Management in a DBMS

Low-level file management serves as a way of abstracting device drivers, however additional organisation at higher levels is often more useful, particularly in DBMSs.

In a DBMS, most data consists of data that has a similar structure and function, for example records concerning personnel.

DBMS file management aims to arrange and format records on disk such that space and access costs are minimised.

DBMS Disk Space Management

There are two main approaches to DBMS disk space management.

1. The DBMS depends on the OS file system. The whole DBMS is allocated one or more OS files. The DBMS is then responsible for space management in the OS files

2. The DBMS does not depend on the OS facilities and manages the disks and memory itself

In either case, one of the tasks is to keep track of free blocks. This is done by bit maps or linked lists.

DBMS vs. OS File system

If the OS does disk space and buffer management, why not let the OS manage these tasks for the DBMS?

- Differences in OS support
- Some limitations (e.g. files can't span disks)
- Buffer management in DBMS requires ability to
  - Pin page in buffer pool, force a page to disk (as discussed earlier)
  - Adjust the replacement policy, and pre-fetch pages based on access patterns in typical DB operations (different *page reference patterns*)

We discuss DBMS file organisation in the next lecture.

Databases and Files

A database may contain many files.

A file itself is stored as a set of blocks on the disk, that is, the data is made up of many logical blocks.

How do we keep track of the blocks belonging to a file and how do we allocate blocks to a file?

- By allocating contiguous blocks, and keeping track of first block and total number of blocks
- By using a linked list
- By using a directory of blocks. Note that we could allocate clusters of contiguous blocks also in this case.

We discuss these in detail next.

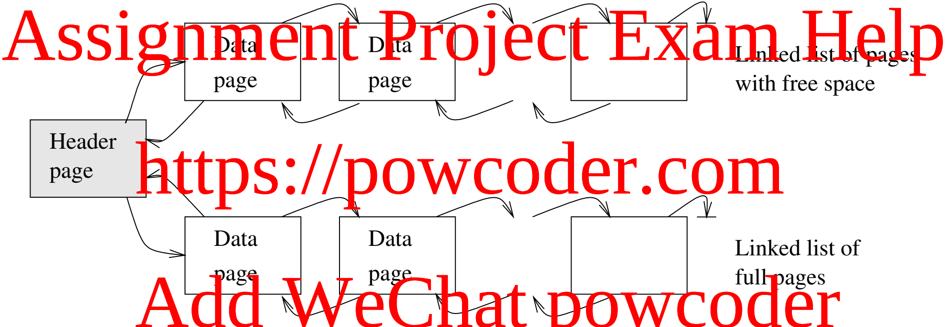
Unordered (Heap) Files

- Simplest structure contains *records* in no particular order (more on records later and file structures in the next lecture)
- As the file grows and shrinks, disk blocks are allocated and de-allocated

- To support *record-level* operations, we must:
  - keep track of blocks in a file
  - keep track of free space on blocks
  - keep track of *records* on blocks

(There are many alternative solutions to these problems.)

Heap File Linked List Approach



Data page

Data page

Linked list of pages with free space

Header page

Data page

Data page

Linked list of full pages

One alternative is the heap file approach. Here, we maintain a linked list of blocks that have free space—those where data can be inserted—and a second list of blocks that are full.

Heap File Linked List Approach...

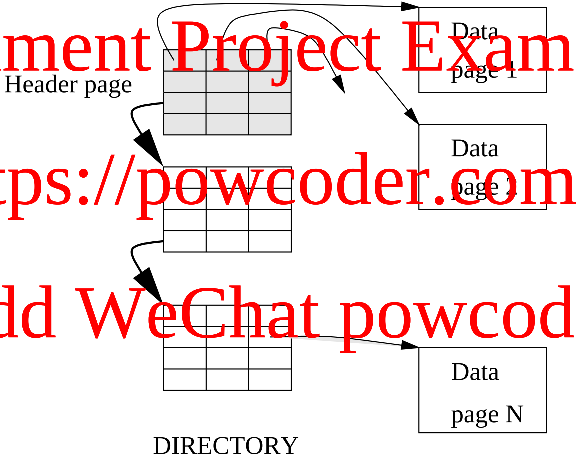
If new blocks are required, a request is made of the buffer manager, and the new block is then added to the list of blocks in the file (probably at first as a block with free space).

Deletion is simple.

A disadvantage of this approach is that—because of internal fragmentation of the blocks—almost all blocks will be on the list of blocks with free space.

(The directory-based heap file solves the problem.)

Heap File Block Directory Approach



Heap File Block Directory Approach...

- The directory entry for a block can include the number of free bytes on the block
- The directory is a collection of blocks, and the linked list implementation is just one alternative
- The directory can be efficiently searched and managed; directory blocks can be allocated and deallocated as needed.
- Directories contain little information and are therefore likely to be small compared to the data blocks

Summary: Disks and Files

In this lecture we have discussed:

- Memory hierarchy
- Disks and disk characteristics
- Buffers and caching
- Buffer management in a DBMS
- Heap file organisation

Next lecture, we will cover data, records, and page formats.