Study of data storage in a database management systems
We shall learn the basic techniques for managing data within the computer
There are two issues we must address which are related to how a DBMS deals with very large amounts of data efficiently:

- How does a computer system store and manage very large volumes of data?
- What representations and data structures best support efficient manipulations of this data?
Today

- Hardware: Disks
- Access Times
- Optimizations
- Other Topics:
  - Storage costs
  - Using secondary storage
  - Disk failures
Hardware

DBMS

Data Storage
Data Storage

- How does a DBMS store and access data?
  - main memory (fast, temporary)
  - disk (slow, permanent)
- How do we move data from disk to main memory?
  - buffer manager
- How do we organize relational data into files?
DBMS stores information on ("hard") disks.

This has major implications for DBMS design!
- **READ**: transfer data from disk to main memory (RAM).
- **WRITE**: transfer data from RAM to disk.
- Both are high-cost operations, relative to in-memory operations, so must be planned carefully!
Why Not Store Everything in Memory?

- Relatively high cost
- Main memory is not persistent (volatile)
  - We want data to be saved between runs. (Obviously!)
- Data Size > Memory Size > Address Space
- Note: many “In-memory” databases are available, and used increasingly for applications with small storage requirements and as memory sizes increase
Typical Storage Hierarchy

- CPU Registers - temporary variables
- Cash - Fast copies of frequently accessed memory locations
- Main memory (RAM) for currently used “addressable” data.
- Disk for main database (secondary storage)
- Tapes for archiving older versions of the data (tertiary storage)
A typical computer system has several different components in which data may be stored.
The use of secondary storage is one of the important characteristics of a DBMS.

To motivate many of the ideas used in DBMS implementation, we must examine the operation of disks in detail.
Disks

- Secondary storage device of choice
- Main advantage over tapes: **random access** vs. **sequential**
  - Sequential: read the data contiguously
  - Random: read the data from anywhere at any time
- Data is stored and retrieved in units called disk blocks or pages
- Retrieval time depends upon the location of the disk
  - Therefore, relative placement of pages on disk has major impact on DBMS performance! **Why?**
Components of a Disk

- **Platter**: circular hard surface on which data is stored by inducing magnetic changes
  - **Platters** are 2-sided and magnetic
- **Platters** rotates (7200 RPM - 15000 RPM)
  - **RPM** (Rotations Per Minute)
- All disk heads move at the same time (in or out)
Platter has circular tracks
Tracks are divided into sectors
Sector: the unit of write operation for a disk
However:
  Sector is too small to be efficient
  Computer systems read/write a block (multiple sectors) at once.
A block (page) consists of one or more multiple contiguous hardware sectors
  Between main memory and disk the data is moved in blocks
  Block size: 4K-64K bytes
Gaps are non-magnetic and used to identify the start of a sector
Top View of a Platter

gap

sector

track
**Terminology: cylinder**

- **Cylinder**: all tracks at the same distance from the center/tracks that are under the heads at the same time
- Disk head does not need to move when accessing (read/write) data in the same cylinder
Disk Storage Characteristics

- # Cylinders = # tracks per surface (platter)
  - e.g., 10 tracks ⇒ 10 cylinders and we can refer to them cylinder zero to cylinder nine
- # tracks per cylinder = # of heads or \(2 \times \) # platter
- Average # sectors per track
- bytes per sector
- ⇒ disk capacity/size
Today

- Hardware: Disks
- Access Times
- Optimizations
- Other Topics:
  - Storage costs
  - Using secondary storage
  - Disk failures
The time taken between the moment at which the command to read a block is issued and the time that the contents of the block appear in main memory is called the *latency of the disk*.

The *access time* is also called the *latency of the disk*.
Basic operations:
- **READ**: transfer data from disk to buffer
- **WRITE**: transfer data from buffer to disk

Note that blocks can be read or written only when:
- The heads are positioned at the cylinder containing the track on which the block is located, and
- The sectors contained in the block move under the disk head as the entire disk assembly rotates.
Accessing the Disk

access time = seek time + rotational delay + transfer time + other delay

- **Other Delays:**
  - CPU time to issue I/O
  - Contention for controller
    - Different programs can be using the disk
  - Contention for bus, memory
    - Different programs can be transferring data
  - These delays are negligible compared to Seek time + rotational delay + transfer time
  - “Typical” Value: 0
Accessing the Disk

access time = seek time + rotational delay + transfer time

- **Seek time**: time to move the arm to position disk head on the right track (position the read/write head at the proper cylinder)
- **Seek time** can be 0 if the heads happen already to be at the proper cylinder.
- If not, the heads require some minimum time to start moving and to stop again, plus additional time that is roughly proportional to the distance traveled.
- The **average seek time** is often used as a way to characterize the speed of the disk.
Accessing the Disk

access time = seek time + rotational delay + transfer time

- rotational delay: time to wait for sector to rotate under the disk head
- i.e., wait for the beginning of the block
On the average, the desired sector will be about half way around the circle when the heads arrive at its cylinder.

Average rotational delay is time for \( \frac{1}{2} \) revolution

Example: Given a total revolution of 7200 RPM
- One rotation = \( \frac{60\text{s}}{7200} \) = 8.33 ms
- Average rotational latency = 4.16 ms
Accessing the Disk

**access time** = seek time + rotational delay + transfer time

- **data transfer time**: time to move the data to/from the disk surface
- **Transfer time** is the time it takes the sectors of the block and any gaps between them to rotate past the head.
- Given a transfer rate, the transfer time = \( \frac{\text{Amount data transferred}}{\text{transfer rate}} \)
- Transfer Rate: \( \# \) bits transferred/sec
Steps to access data on a disk

1. Move the disk heads to the desired cylinder
   - Time to seek a cylinder = seek time
Steps to access data on a disk

2. Wait for the desired sector to arrive under the disk head
   - Time to wait for a sector = rotational delay
3. Transfer the data from sector to main memory (through the disk controller)
Accessing the Disk

- Seek time and rotational delay dominate.
- Key to lower I/O cost: reduce seek/rotation delays!
Arranging Blocks on Disk

- So far: One (Random) Block Access
- What about: Reading “Next” block?
- Blocks in a file should be arranged sequentially on disk (by “next”) to minimize seek and rotational delay.
- **Next** block concept:
  - blocks on same track, followed by
  - blocks on same cylinder, followed by
  - blocks on adjacent cylinder
- For a sequential scan, pre-fetching several blocks at a time is a big win.
If we do things right

- (e.g., Double Buffer, Stagger Blocks...)
- Time to get blocks should be proportional to the size of blocks, and the seek time and rotational latency thus become trivial
- time to get block = \( \frac{\text{Block size}}{\text{transfer rate}} \) + Negligible
- Negligible:
  - skip gap
  - switch track
  - once in a while, next cylinder
Rule of Thumb

- Random I/O: Expensive
- Sequential I/O: Much less
The process of writing a block is, in its simplest form, quite similar to reading a block

... unless we want to verify!

need to add (full) rotation $+ \frac{\text{Block size}}{\text{transfer rate}}$
To Modify a Block?

It is not possible to modify a block on disk directly. Rather, even if we wish to modify only a few bytes, we must do the following:

1. Read Block into Memory
2. Modify in Memory
3. Write Block
4. [Verify?]
SSD (SOLID STATE DRIVE)

- SSDs use flash memory
- No moving parts (no rotate/seek motors)
  - eliminates seek time and rotational delay
  - very low power and lightweight
- Data transfer rates: 300-600 MB/s
- SSDs can read data (sequential or random) very fast!
- Small storage (0.1 – 0.5× of HDD)
- expensive (20× of HDD)
- Writes are much more expensive than reads (10×)
- Limited lifetime
  - 1-10K writes per page
  - the average failure rate is 6 years
Today

- Hardware: Disks
- Access Times
- Optimizations
- Other Topics:
  - Storage costs
  - Using secondary storage
  - Disk failures
Effective ways to speed up disk accesses:

- Disk Scheduling Algorithms
- Pre-fetch (a.k.a. Double buffering)
- Arrays (RAID)
- Mirrored Disks
- On Disk Cache
Situation: Have many read/write requests at any one moment in time

Question: **Service policy**: In which order do you process (service) the requests?

- The order in which you service the disk operations can affect the performance

Naïve service (but fair): **First Come First Serve**

- Fairness but inefficient (e.g. zig-zag read pattern)

Commonly used disk scheduling algorithm: **the “elevator” algorithm**

- Elevator scheduling for a disk:
  - The disk head sweeps in-and-out (like an elevator)
  - When the disk head is on a cylinder $k$:
  - Disk will service all requests for that cylinder before moving to the next cylinder

- Efficiency but unfair
Pre-fetching (Double Buffer)

- Software level solution: pre-fetching: The program can use pre-fetching to reduce processing time.
- In some scenarios, we can predict the order in which blocks will be requested from disk by some process.
- Pre-fetching (double buffering) is the method of fetching the necessary blocks into the buffer in advance
- Requires enough buffer space
- Speedup factor up to $n$, where $n$ is the number of blocks requested by a process
Double Buffering Algorithm

Problem
- Have a File
  - Sequence of Blocks B1, B2, ...
- Have a Program
  - Process B1
  - Process B2
  - Process B3
  - ...
Single Buffer Solution (Naïve Solution)

1. Read B1 → Buffer
2. Process Data in Buffer
3. Read B2 → Buffer
4. Process Data in Buffer
Let:

- \( P \) = time to process/block
- \( R \) = time to read in 1 block
- \( n \) = number of blocks

1. Read B1 \( \rightarrow \) Buffer \( \Rightarrow \) \( R \)
2. Process Data in Buffer \( \Rightarrow \) \( P \)
3. Read B2 \( \rightarrow \) Buffer \( \Rightarrow \) \( R \)
4. Process Data in Buffer \( \Rightarrow \) \( P \)

Time to process \( n \) block = \( n(P+R) \)
Double Buffering Solution

- The program allocates two buffers to process data from a file
- Data is read in a buffer:
  - When buffer is full, program processes the data. And at the same time, more data is read in the other buffer
- Rotate buffers when done processing data in buffer
Double Buffering Solution

Memory:

Disk: A B C D E F G

Read block 1

process
Double Buffering Solution

Memory:

Disk:

Process block 1

AND read block 2 simultaneously

done
Double Buffering Solution

Memory:

Disk:

AND read block 3 simultaneously

Process block 2

done
Double Buffer Solution

Let:

- $P =$ time to process/block
- $R =$ time to read in 1 block
- $n =$ # blocks
- Say $P \geq R$

What is processing time?

- Double buffering time $= R + nP$
- Single buffer time $= n(R + P)$
Using disk array to accelerate disk access

- Why use multiple disks
  - Multiple disks $\rightarrow$ multiple disk heads
  - Multiple outputs $=$ Increased data rate
Techniques: multiple disks

- **Block Striping**
  - Store blocks of a file over multiple disks

- **Mirror disk**
  - Store the same data on multiple disks
  - Mirrored disks contain identical content
  - Read operation: \( n \) times as fast
  - Write operation: about the same as 1 disk

- **RAID**
  - Redundant Array of Independent (inexpensive) Disks
Disk Failures

We consider ways in which disks can fail and what can be done to mitigate these failures:

- Intermittent read failure (Cause: power fluctuations/failure)
- Intermittent write failure (Cause: power fluctuation/failure)
- Media decay (Disk surface worn out)
- Permanent failure (Disk crash)
Coping with Read/Write Failures

- **Detection**
  - Read (verify) after writing data
  - Better: Use checksum
- **Correction**
  - Redundancy
Coping with media decay

- Disk has a number of spare blocks
- When writing a block fails for \( n \) times
  - Mark block as bad
  - Replace block with one of the spare blocks
Coping with Disk Crash

- Different ways to achieve redundancy
  - Exact copy (mirror)
  - RAID
Megatron 747 Disk (old)

Example

- Rotate at 3600 RPM
- Only 1 surface
- 16 MB usable capacity (usable capacity excludes the gaps)
- 128 cylinders
- seek time:
  - average = 25 ms.
  - adjacent cylinders = 5 ms.
- 1 KB block = 1 sector
- 10% overhead between blocks
  - gaps represent 10% of the circle and
  - sectors represent the remaining 90%
Megatron 747 Disk (old)

- 1 KB blocks = sectors
- 10% overhead between blocks
- capacity = 16 MB = \( (2^{20}) \times 16 = 2^{24} \)
- \# cylinders = 128 = \( 2^7 \)

bytes/cylinder = \( \frac{\text{total capacity}}{\text{total \# cylinders}} = \frac{2^{20} \times 16}{128} = \frac{2^{24}}{2^7} = 2^{17} = 128\text{KB} \)

#blocks/cylinder = \( \frac{\text{capacity of each cylinder}}{\text{size of block}} = \frac{128\text{KB}}{1\text{KB}} = 128 \)
Megatron 747 Disk (old)

- 3600 RPM → 60 revolutions/sec → 1 rev. = 16.66 msec.
  
  One track:
  
  Time over useful data = $16.66 \times 0.9 = 14.99$ ms
  
  Time over gaps = $16.66 \times 0.1 = 1.66$ ms
  
  Transfer time for 1 block = $\frac{14.99}{128} = 0.117$ ms
  
  Transfer time for 1 block+gap = $\frac{16.66}{128} = 0.13$ ms
Megatron 747 Disk (old)

- Access time \( (T_1) \) = Time to read one random block
- \( T_1 = \text{seek} + \text{rotational delay} + \text{transfer time for 1 block} \)
- \( T_1 = 25 + \frac{16.66}{2} + 0.117 = 33.45 \text{ ms.} \)
- Why we did not use the time it takes to transfer 1 block+gap here?
Suppose OS deals with 4 KB blocks

Access time = $T_4 = 25 + \frac{16.66}{2} + 0.117 \times 1 + 0.13 \times 3 = 33.83$ ms

Compare to $T_1 = 33.45$ ms

Q) The time to read a full track is?
Summary

- Secondary storage, mainly disks
- I/O times
- I/Os should be avoided, especially random ones
Chapter 2: data storage in Assignments & Projects/reading folder, except Sections: 2.3.3, 2.3.4, 2.3.5, 2.4.4, 2.5.4, 2.6
File and System Structure