# CSE 411 / L-01 / 10.07.2024 /

Attendance ⇒ 10%.

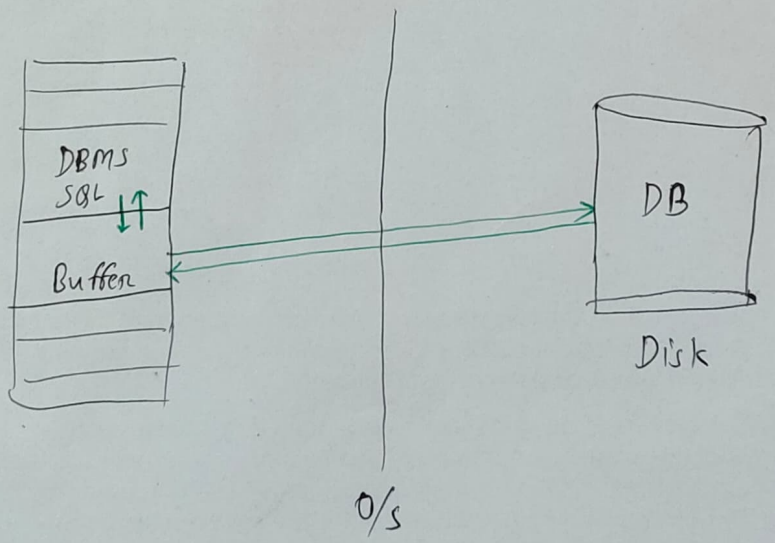
Quiz ⇒ 2
Assignment = 1
} Best 2 ⇒ 20%.

Midterm ⇒ 30%.

Final ⇒ 40%.

Book ⇒ system concept 7th edition.

## Disk Based DBMS System




- Data read/write through Buffer.
  - SQL runs on Buffer.
- SQL first search on Buffer, if not found, take it into buffer then provide output.
- Disk access time in milisecond.
  But memory access time in nano second.

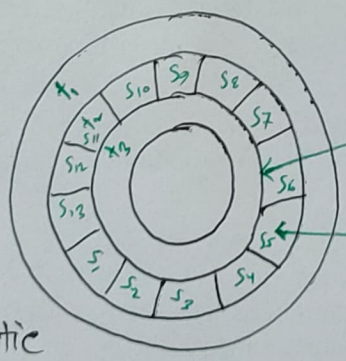
## Storage architecture

⇒ two type of storage.

- Volatile ⇒ primary memory
  - content will be erase when power off.
- Non-Volatile ⇒ secondary memory
  - content will remain..

## Storage Hierarchy - Slide-12

- Top 4 memory type are related to the database.
  - Direct addressable
- Bottom two are not possible to access directly or randomly. These need sequential access.
  - sequential addressable.




Track ⇒ normally 50 ... 100 k

Sector = normally 512 Byte

Magnetic Disk

⇒ Sector per track = 500 ... 1000

- outer track have more sector than inner track

Student (ID, Name, street, city)

Address : Track - 500 & sector - 200

Query : SELECT *
FROM student

Current head position in track-1.
No data in Buffer.

⇒ What will be the access time?

⇒ Access time = start from "SQL issue" and end in "start of data transfer."

Access time = Head Reposition time from track 1 to 500
+
time to place the head at sector - 200
(need to rotate the disk.)

Track reposition time = seek time (4 - 10 milliseconds)
Sector reposition time = Rotational latency (4-11 milliseconds)
(5-20 milliseconds)

Data transfer rate: depends on handware manufacturer.

⇒ 25 - 200 MBps
25 → inner tracks
200 → outer tracks

Average seek time and rotational latency will be given, calculate in percentage.

If average given, then max is = 2 × average.

As we know, average = $\frac{\text{best} + \text{worst}}{2} = \frac{0 + \text{max}}{2}$ ⇒ max = 2 · ave

L-02 / 15.07.2024 /

# RAID

## Redundant Arrays of Independent Disks:

- use of multiple disk in parallel provides high capacity and high speed with high reliability.

Block Size
⇒ 4 kB, 8 kB, 16 kB.

## Bit level striping:

- split the bits of each byte accross multiple disks. one bit per disk
- seek/access time worse than for a single disk.
- Not so efficient for large data.

## Block level striping:

i-block will go to $(i \bmod n) + 1$ (disk number)

## Raid level-0:

- No redundant data
- only one copy of data, if a disk fall then data will be lose.
- But provide high performance.

## Raid level - 1:

- Best write performance
- two copy of data, like a mirror disk

Given, disk size = 4 TB

How many disk will be needed to ~~bar~~ store 24 TB effective storage?

RAID-0 ⇒ 6×

RAID-1 ⇒ 12× (6+6)

## RAID-5: Block striping distributed parity

- parity bit can help us to recover a data.
  - → odd parity
  - → even parity
- support one disk failure.

Recovery mechanism and updating mechanism.

## L-03 / 02.09.2024

### Indexing

Student

| ID | Name | Street | City |
|---|---|---|---|
| 1 | Abdullah | | Comilla |
| 2 | Abid | | Dhaka |
| ⋮ | ⋮ | | ⋮ |
| 3000 | Abid | | Rajshahi |
| ⋮ | ⋮ | | ⋮ |
| 10000 | Anif | | Dhaka |
| ⋮ | ⋮ | | ⋮ |
| 20000 | Abid | | Dhaka |

Query:

```
CREATE INDEX Name
ON Student (Name)
```

| Search key | Pointer |
|---|---|
| Abdullah | 1 |
| Abid | multiple |
| Anif | 10,000 |

Sorted

Bucket

| |
|---|
| 2 |
| 3000 |
| 20000 |
| |

## Syntax:

```
CREATE INDEX index_name
ON table_name (column1, column2 ...)
```

- Index file must be ~~ordered~~ ordered/sorted.
- Two kinds of indices
  - Ordered
  - Hash

## Index Evaluation Metrics

⇒ Insertion Time:

- if there are 50 attribute in a table then we need to insert the search key on 50 index first then we can insert in database.

⇒ Deletion Time:

- same as insertion

⇒ Access Time: depends on type.

point access (==)

range access (<, >)

- Space overhead for large table with so many attribute.

# [L-04/04.02.2024]

## Indexing

Access type:

- point access ⇒ (=)
- range access ⇒ (<, >)

### Primary Index / Clustering Index:

- order of index file = order of original data

⇒ not necessarily the primary key.

### Secondary Index:

- order of index file ≠ order of original data

### Dense Index:

one index/search key = one data

- index record appears for every search-key value in the file.

**Query-1:**

```
SELECT *
FROM Instructor
WHERE ID = 83821
```

⇒ Point access

**Query-2:**

```
SELECT *
FROM Instructor
WHERE ID > 58583
```

⇒ Range access

⇒ Let one tuple/row/data = 1 Block

then, instructor relation size = 12 blocks.

Question - 1:

- Explain how Query - 1 and Query - 2 will be processed without index file?

Question - 2:

- Explain how Query - 1 and Query - 2 will be processed with index file?

Seek time = movement time of disk head.

For Query - 1:

- cost without index:

number of seek = 1

number of block transfer = 11

- cost with index: → already in memory

number of seek = 1

number of block transfer = 1

For Query - 2:

- cost without index:

number of seek = 1

number of block transfer = 12

- cost with index:

number of seek = 1

number of block transfer = 4

What will happen if the search key is a non-primary column / or secondary index?

Primary index, secondary index query processing time - important for the exam.

## Primary Index:

order of index file = order of dataset

ordered index → Dense (one to one)
→ Sparse (one to many)

↳ applicable only when the dataset are sorted physically.
- primary index.
- means, all sparse index are primary index.

Query Cost:

sparse > Dense

**Query-3:**

```
SELECT *
FROM Instructor
WHERE ID = 76543
```

**Question-3:** find out the cost using sparse index?

⇒ seek = 1
block transfer = 5

What will happen when the search key exist on the sparse index directly?

What will happen in the case of range access?

L-05/ 09. 09. 2024

## Secondary Indices

- secondary index have to be dense.

**Query-1:**

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

Cost:
With index:
seek = 2
Block transfer = 2
Without index:
seek = 1
Block transfer = 12

**Query -2:**

```
SELECT *
FROM Instructor
WHERE Salary >= 65000
```

Cost:
With index:
seek = 9
Block transfer = 9
without index:
seek = 1
Block transfer = 12

- Seek time is almost 10x of Block transfer time.

## $B^+$ tree index file

- Three types of nodes:
  i. Root
  ii. Internal
  iii. Leaf

Number of pointe = $n$ → usually it is 100

Number of maximum search key in a block/node = $n-1$

- minimum = $\lceil \frac{n-1}{2} \rceil$

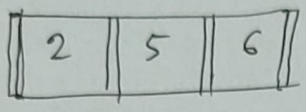
n = 4

Data = 2, 5, 6, 7, 10

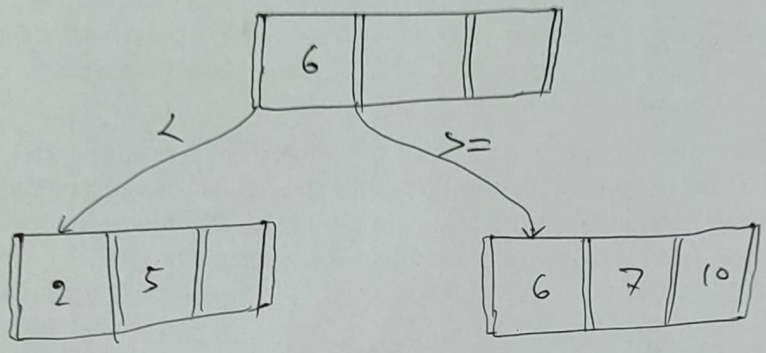
min = 2

max = 3

⇒




⇒




Root & internal, there will be no duplicate search-key. But all the search-key will be available in the leaf node And leaf nodes are in sorted order.

Advantage and Disadvantage

slide - 34

Query - 3:

```
SELECT *
FROM Instructor
WHERE Name > "Crick"
      and Name < "kim"
```

cost:

seek = 9

Block Transfer = 9

## Query-4:

```
SELECT *
FROM Instructor
WHERE Name = 'Singh'
```

Cost:

seek = 4

Block Transfer = 4

Quiz-1 upto this 30.09.2024

L-06 / 11.09.2024 /

## Query Processing & Optimization

Algebra:

$\sigma$ → Select

$\Pi$ → Project

## Query-1:

```
SELECT ID
FROM Student
```

Algebra:

$\Pi_{ID}(Student)$

## Query-2:

```
SELECT *
FROM Student
WHERE CGPA > 3.0
```

Algebra:

$\sigma_{CGPA > 3.0}(Student)$

$\sigma$ ⇒ for all column.

$\Pi$ ⇒ for some specific column.

## Basic Steps | Slide-3

i. Parse and Translate to relational algebra expression (Syntax analyzer)

ii. Optimize the query with the help of statistic info of database.

iii. Evaluate.

If intermediate result is low then the cost will be low.

## Query Evaluation:

⇒ many factors contribute to the time cost

- disk access
- CPU
- network communication

But we need to consider only the response time or resource consumption.

$t_T$ = time to transfer one block

$t_s$ = time for one seek.

Seek time is usually 40-50 times than block transfer time.

A 1 (Linear Search)

- Scan all the block

$\Rightarrow t_s + b_R t_T$

Problem -1: Secondary Index

Size = 2000 blocks

seek = 5 ms

block transfer = 0.1 ms

Query:

SELECT *

FROM student

WHERE ~~GPA~~ CGPA > 3.5

$\Rightarrow$ cost:

$b_R = 2000$

cost = $t_s + b_R t_T$

$= (5 + 2000 \cdot 0.1)$ ms

$= 205$ ms

Problem -2: Primary Index

Query:

SELECT *

FROM Student

WHERE ID < 1801111042

Result = 800 blocks

$\therefore b_R = 800$

$\Rightarrow$ cost = $t_s + b_R t_T$

$= (5 + 800 \cdot 0.1)$ ms

$= 85$ ms

## A2 (clustering index equality on key)

- Primary index — retrieve single record.
- $B^+$ tree
- height $h_i$ given

$$\text{cost} = \underbrace{h_i * (t_s + t_T)}_{\text{index cost}} + \underbrace{(t_s + t_T)}_{\text{data copy cost}}$$

Query -3s

```
SELECT *
FROM student
WHERE ID = 1520303042
```

cost = $h_i * (t_s + t_T) + (t_s + t_T)$

## A3 (clustering index, equality on nonkey)

- secondary index
- retrieve multiple records, consecutive

$$\text{cost} = h_i * (t_s + t_T) + t_s + b * t_T$$

($b$ → number of block containing matching record.)

Query:

```
SELECT *
FROM Student
WHERE City = "Uttara"
```

Number of student = 400 (uttara)
each record size = 400 bytes
each block size = 4000 bytes
record per block = 4000/400 = 10
∴ total block need to transfer = 400/10 = 40

$B^+$ tree height = 2

∴ cost = $h_i * (t_s + t_T) + t_s + b * t_T$
= $2 * (10 + 0.2) + 10 + 40 * 0.2$ = 38.4 ms

# L-07 / 18.09.2024 /

* A4 (Secondary index, equality on key/non-key)

number of record that matched.

$$\text{cost} \Rightarrow (h_i + n) * (t_s + t_T)$$

$$= \underbrace{h_i * (t_s + t_T)}_{\text{index cost}} + \underbrace{n(t_s + t_T)}_{\text{block transfer, data cost}}$$

* A7 (conjunctive selection using one index)

- apply A4 for each indices and find out the best one.

Problem - 5:

For CGPA

$$\text{cost} = (h_i + n) * (t_s + t_T)$$
$$= (2 + 5) * (10 + 0.2)$$
$$= 71.4 \text{ ms}$$

for City

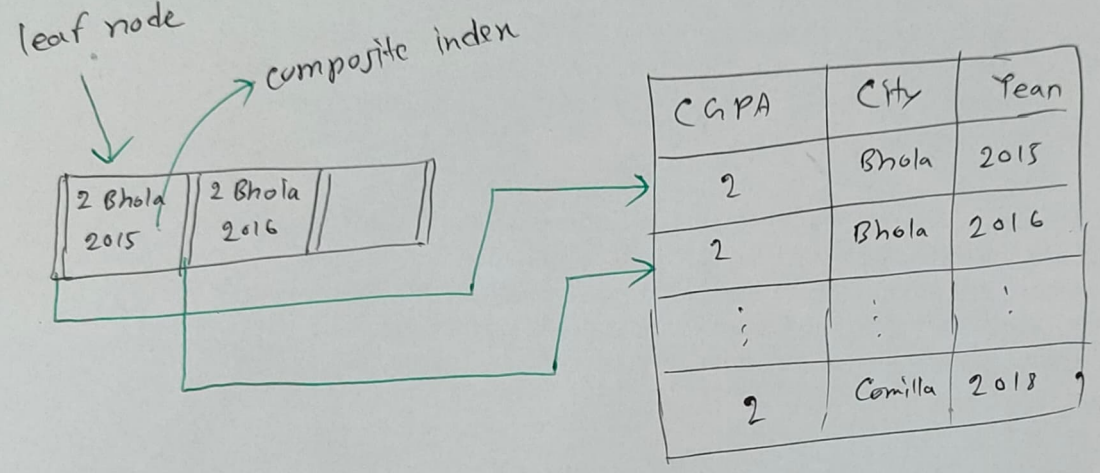
$$\text{cost} = (h_i + n) * (t_s + t_T)$$
$$= (2 + 20) * (10 + 0.2)$$
$$= 224.4 \text{ ms}$$

For Year.

$$\text{cost} = (h_i + n) * (t_s + t_T)$$
$$= (1 + 4000) * (10 + 0.2)$$
$$= 40810.2 \text{ ms}$$

## A8 (conjunctive selection using composite index)

- if composite index available, then it will be faster than others.
- fully optimized, but require more storage.




Query:-

```
SELECT *
FROM Student
WHERE CGPA=2 AND City = "Bhola" AND Year = 2015
```

cost = index cost + data cost

$= (h_i + n) * (t_s + t_T)$

## A9 (Conjunctive selection using intersection of identifiers)

- just use the index ~~att~~ and all pointer to find out the data location.
- we do not need to transfer data to intersect.
- we can simply intersect the pointer of the buckets of index.

Cost = cost of each index to transfer pointer list

+

data cost

Problem -7:

$$cost = \left(\Sigma(h_i) + n\right) * \left(t_S + t_T\right)$$

$$= (2+2+1+1) * (10+0.2)$$

$$= 61.2 \text{ ms.}$$

## L-08 / 23.09.2024 /

### Recap Algorithm.

- Student (ID, Name, Street, City, CGPA)

  Index given = primary index only (ID)
  = secondary index on (Name, Street, city)

- Query-1:

```
SELECT *
FROM Student
WHERE ID = 1234567
```

Find the algorithm?

⇒ A2

Query -2:

```
SELECT *
FROM Student
WHERE CGPA = 3.5
```

⇒ A1

- CGPA is not in the index list, so we need to perform linear search

Query - 3:

```
SELECT *
FROM Student
WHERE ID > 2345678
      AND Name = "Abdullah"
```

⇒ A7, A9

- composite index not available, so A8 not possible.

## Sorting

We may build an index on the relation, and then use the index to read the relation in sorted order. May lead to one disk block access for each tuple.

- we can copy ~~the~~ or transfer the relation to memory and apply quick-sort,
- But if a relation do not fit in the memory, what to do?
  - external sort-merge is a good solution for this problem.

## External Sort - Merge:

N = number of runs

M = memory size in terms of run.

possible to merge = M - 1 runs (→ store the merge result.)

Given,

N = 90

M = 11

∴ Run required = $\frac{90}{11-1} = \frac{90}{10} = 9$ times

Then, we can merge the 9 runs at once

So total Run time = 9 + 1 = 10 times

And the last one will be showed as output to the terminal, no need to store in the disk.

Given,

Number of block = 900

Memory = 10 block

Number of Runs = $\left\lceil \frac{900}{10} \right\rceil = 90$

⇒

Total block = $b_R$

memory size = M

Number of initial run = $\left\lceil \frac{b_R}{M} \right\rceil$

1st merge, number of runs = $\left\lceil \left\lceil \frac{b_R}{M} \right\rceil / M-1 \right\rceil$

2nd merge, number of runs = $\left\lceil \left\lceil \frac{b_R}{M} \right\rceil / M-1 / M-1 \right\rceil$

$= \left\lceil \left\lceil \frac{b_R}{M} \right\rceil / (M-1)^2 \right\rceil$

Therefore,

Number of merge pass = $\left\lceil \log_{M-1} \left\lceil \frac{b_R}{M} \right\rceil \right\rceil$

$P = m^n$

$n = \log_m P$

L-09 (25.09.2024)

| | | |
|---|---|---|
| $R_1$ | $R_1'$ | $R_1''$ |
| $R_2$ | $R_2'$ | $R_2''$ |
| $R_3$ | $R_3'$ | $R_3''$ |
| ⋮ | ⋮ | ⋮ |
| $R$ | $R_{\left(\frac{N}{M-1}\right)}$ | $R_{\frac{N}{(M-1)^2}}$ |
| $\frac{N}{M-1}$ | $\frac{\lceil \frac{B_R}{M} \rceil}{M-1}$ | $\frac{\lceil \frac{B_R}{M} \rceil}{(M-1)^2}$ |

(x) Number of merge pass = $\log_{M-1} \left\lceil \frac{B_R}{M} \right\rceil$

(x) Total Blocks = $B_R$

✳ Block transfer for run creation = $2 \cdot B_r$

✳ Total Block Transfer = $2B_r + \underbrace{2B_r * \log_{(m-1)} \lceil \frac{B_r}{m} \rceil}_{\text{Merge Pass}} - B_r$

$2B_r$ → Run Creation

$B_r$ → Last result will not store to the disk.

∴ Total Seek = Run Creation + Merge Pass

$$= 2 \lceil \frac{B_r}{m} \rceil + 2B_r \lceil \log_{m-1} \frac{B_r}{m} \rceil - B_r$$

## ✳ Join Operation

- Nested Loop
- Block nested loop

✳ Student (ID, Name, Street, City, CGPA)

Takes (ID, CourseID, Semester, Year, Grade)

⇒ Find students and corresponding course information.

∴ SQL -

```
SELECT *
FROM student INNER JOIN Takes ON student.ID =
                                 Takes.ID
```

Algebra: Student ⋈ Takes ⇒ What will be the cost

⇒ Find out cost of Takes ⋈ Student ?

⊛ Will they takes different cost? why?

⇒ Estimated block transfer =

= number of tuples in first relation * number of block in second relation + number of block in first relation

$= n_r * b_s + b_r$

r ⋈ s

⇒ Number of seek

= number of tuples in first relation + number of block in first relation

$= n_r + b_r$

Best Case
B.T. = $b_s + b_r$
seek = 2

Therefore,

Student ⋈ Taks

block transfer = 20,000 * 400 + 2,000

= 80,02,000

seek = 20,000 + 2,000

= 22,000

Takes ⋈ Student

Block Transfer = 40,000 * 2,000 + 400 = 8,00,00,400

seek = 40,000 + 400 = 40,400