



CS 525: Advanced Database Organization



01: Introduction

Boris Glavic


Slides: adapted from a [course](#) taught by [Hector Garcia-Molina](#), Stanford InfoLab

CS 525

Notes 1 - Introduction
1 IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

1

Advanced Database Organization?


- =Database Implementation
- =How to implement a database system
- ... and have fun doing it ;-)

CS 525

Notes 1 - Introduction
2 IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

2

Isn't Implementing a Database System Simple?

Relations \implies Statements \implies Results


CS 525

Notes 1 - Introduction
3 IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

3

Introducing the MEGATRON 3000

Database Management System


- The latest from Megatron Labs
- Incorporates latest relational technology
- UNIX compatible

CS 525

Notes 1 - Introduction
4 IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

4

Megatron 3000 Implementation Details

! First sign non-disclosure agreement !


CS 525

Notes 1 - Introduction
5 IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

5

Megatron 3000 Implementation Details

- Relations stored in files (ASCII)
e.g., relation R is in /usr/db/R

```
Smith # 123 # CS
Jones # 522 # EE
:
```

CS 525

Notes 1 - Introduction
6 IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

6

Megatron 3000

Implementation Details

- Directory file (ASCII) in /usr/db/directory

```

R1 # A # INT # B # STR ...
R2 # C # STR # A # INT ...
.
.
.

```

CS 525
Notes 1 - Introduction
7

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

7

Megatron 3000

Sample Sessions

```

% MEGATRON3000
Welcome to MEGATRON 3000!
&
:
:
& quit
%

```

CS 525
Notes 1 - Introduction
8

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

8

Megatron 3000

Sample Sessions

```

& select *
from R #

  Relation R
  A      B      C
SMITH  123   CS
&

```

CS 525
Notes 1 - Introduction
9

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

9

Megatron 3000

Sample Sessions

```

& select A,B
from R,S
where R.A = S.A and S.C > 100 #

  A      B
123   CAR
522   CAT
&

```

CS 525
Notes 1 - Introduction
10

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

10

Megatron 3000

Sample Sessions

```

& select *
from R | LPR #
&

```

Result sent to LPR (printer).

CS 525
Notes 1 - Introduction
11

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

11

Megatron 3000

Sample Sessions

```

& select *
from R
where R.A < 100 | T #
&

```

New relation T created.

CS 525
Notes 1 - Introduction
12

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

12

Megatron 3000

- To execute “`select * from R where condition`”:
 - (1) Read dictionary to get R attributes
 - (2) Read R file, for each line:
 - (a) Check condition
 - (b) If OK, display

CS 525



Notes 1 - Introduction

13

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

13

Megatron 3000

- To execute “`select * from R where condition | T`”:
 - (1) Process select as before
 - (2) Write results to new file T
 - (3) Append new line to dictionary

CS 525



Notes 1 - Introduction

14

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

14

Megatron 3000

- To execute “`select A,B from R,S where condition`”:
 - (1) Read dictionary to get R,S attributes
 - (2) Read R file, for each line:
 - (a) Read S file, for each line:
 - (i) Create join tuple
 - (ii) Check condition
 - (iii) Display if OK

CS 525



Notes 1 - Introduction

15

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

15

What's wrong with the Megatron 3000 DBMS?

CS 525



Notes 1 - Introduction

16

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

16

What's wrong with the Megatron 3000 DBMS?

- Tuple layout on disk
 - e.g., - Change string from 'Cat' to 'Cats' and we have to rewrite file
 - ASCII storage is expensive
 - Deletions are expensive

CS 525



Notes 1 - Introduction

17

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

17

What's wrong with the Megatron 3000 DBMS?

- Search expensive; no indexes
 - e.g., - Cannot find tuple with given key quickly
 - Always have to read full relation

CS 525



Notes 1 - Introduction

18

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY



18

What's wrong with the Megatron 3000 DBMS?

- Brute force query processing

e.g., `select *`
`from R,S`
`where R.A = S.A and S.B > 1000`

- Do select first?
- More efficient join?



CS 525  Notes 1 - Introduction 19  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

19

What's wrong with the Megatron 3000 DBMS?

- No buffer manager



e.g., Need caching

CS 525  Notes 1 - Introduction 20  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

20

What's wrong with the Megatron 3000 DBMS?

- No concurrency control



CS 525  Notes 1 - Introduction 21  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

21

What's wrong with the Megatron 3000 DBMS?

- No reliability

e.g., - Can lose data
- Can leave operations half done



CS 525  Notes 1 - Introduction 22  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

22

What's wrong with the Megatron 3000 DBMS?

- No security

e.g., - File system insecure
- File system security is coarse



CS 525  Notes 1 - Introduction 23  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

23

What's wrong with the Megatron 3000 DBMS?

- No application program interface (API)


e.g., How can a payroll program get at the data?

CS 525  Notes 1 - Introduction 24  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

24

What's wrong with the Megatron 3000 DBMS?


- Cannot interact with other DBMSs.

CS 525  Notes 1 - Introduction 25 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

25

What's wrong with the Megatron 3000 DBMS?


- Poor dictionary facilities

CS 525  Notes 1 - Introduction 26 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

26

What's wrong with the Megatron 3000 DBMS?


- No GUI

CS 525  Notes 1 - Introduction 27 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

27

What's wrong with the Megatron 3000 DBMS?


- Lousy salesman!!

CS 525  Notes 1 - Introduction 28 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

28

Course Overview


- File & System Structure
Records in blocks, dictionary, buffer management,...
- Indexing & Hashing
B-Trees, hashing,...
- Query Processing
Query costs, join strategies,...
- Crash Recovery
Failures, stable storage,...

CS 525  Notes 1 - Introduction 29 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

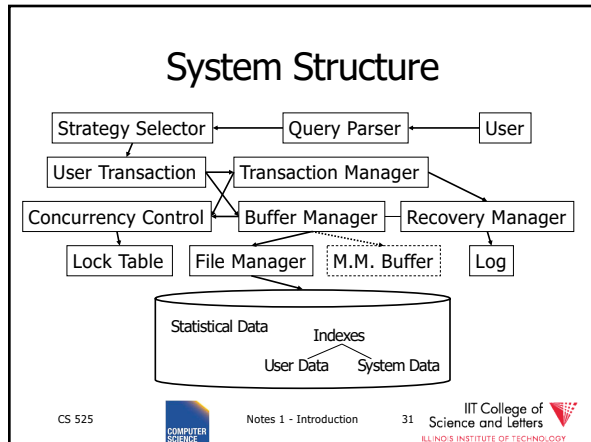
29

Course Overview

- Concurrency Control
Correctness, locks,...
- Transaction Processing
Logs, deadlocks,...
- Security & Integrity
Authorization, encryption,...
- Advanced Topics
Distribution, More Fancy Optimizations, ...

CS 525  Notes 1 - Introduction 30 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY


30



31

Some Terms


- Database system
- Transaction processing system
- File access system
- Information retrieval system

CS 525  Notes 1 - Introduction 32 IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

32

Instructor Info


- **Webpage:** <http://www.cs.iit.edu/~glavic/cs525/>
- **Instructor:** Boris Glavic
 - <http://www.cs.iit.edu/~glavic/>
 - **DBGroup:** <http://www.cs.iit.edu/~dbgroup/>
 - **Office Hours: Wednesdays, 1pm-2pm**
 - <https://meet.google.com/nqv-qnyd-kqf>
- **TA: TBA**

CS 525  Notes 1 - Introduction 33 IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

33

Course Information


- **Time:** Mon + Wed 5:10pm – 8:30pm
- **Live Video Call:**
 - <https://meet.google.com/siw-youu-kme>
- **Live Lectures will be recorded and uploaded to blackboard and to**
 - <https://drive.google.com/drive/folders/1CxZv2sXe8y1dOb4WrNuf30uMkwbiikRd?usp=sharing>

CS 525  Notes 1 - Introduction 34 IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

34

Piazza


- <https://piazza.com/class/kdapgq7az2vg>
- Announcements
- Discussion forum
 - Student - Instructor/TA
 - Student – Student
- ->please join piazza to keep up to date

CS 525  Notes 1 - Introduction 35 IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

35

Workload and Grading

- **Schedule and Important Dates**
 - On webpage & updated there
- **Programming Assignments (50%)**
 - 4 Assignments
 - Groups of 3 students
 - Plagiarism -> 0 points and administrative action
- **Quizzes (10%)**
- **Mid Term (20%) and Final Exam (20%)**

CS 525  Notes 1 - Introduction 36 IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

36

Textbooks

- Elmasri and Navathe , **Fundamentals of Database Systems**, 6th Edition , Addison-Wesley , 2003
- Garcia-Molina, Ullman, and Widom, **Database Systems: The Complete Book**, 2nd Edition, Prentice Hall, 2008
- Ramakrishnan and Gehrke , **Database Management Systems**, 3rd Edition , McGraw-Hill , 2002
- Silberschatz, Korth, and Sudarshan , **Database System Concepts**, 6th Edition , McGraw Hill , 2010

CS 525



Notes 1 - Introduction

37

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

37

Programming Assignments

- 4 assignments one on-top of the other
- Optional 5th assignment for extra credit
- Code has to compile & run on server account
 - Email-ID@fourier.cs.iit.edu
 - Linux machine
 - SSH with X-forwarding
- Source code managed in **git** repository on Bitbucket.org
 - Handing in assignments = submit (push) to repository
 - One repository per student
 - You should have gotten an invitation (if not, contact me/TA)
 - Git tutorials linked on course webpage!

CS 525



Notes 1 - Introduction

38

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

38

Next:

- Hardware

CS 525



Notes 1 - Introduction

39

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

39

CS 525: Advanced Database Organization

02: Hardware



Boris Glavic

Slides: adapted from a [course](#) taught by [Hector Garcia-Molina](#), Stanford InfoLab

CS 525



Notes 2 - Hardware

1



Outline

- Hardware: Disks
- Access Times
- Example - Megatron 747
- Optimizations
- Other Topics:
 - Storage costs
 - Using secondary storage
 - Disk failures

CS 525

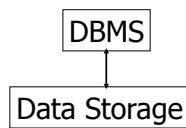


Notes 2 - Hardware

2



Hardware

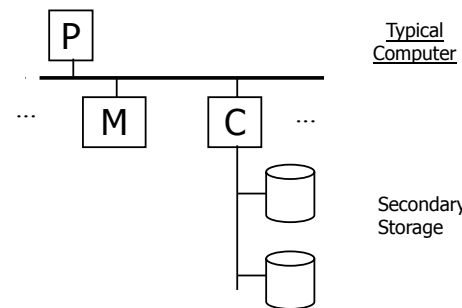


CS 525



Notes 2 - Hardware

3



CS 525



Notes 2 - Hardware

4



Processor

Fast, slow, reduced instruction set, with cache, pipelined...
Speed: 100 → 500 → 1000 MIPS

Memory

Fast, slow, non-volatile, read-only...
Access time: 10^{-6} → 10^{-9} sec.
 $1 \mu\text{s}$ → 1 ns

CS 525



Notes 2 - Hardware

5



Secondary storage

Many flavors:

- Disk: Floppy (hard, soft)
Removable Packs
Winchester
Ram disks
Optical, CD-ROM...
Arrays
- Tape: Reel, cartridge
Robots

CS 525

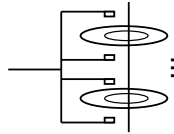


Notes 2 - Hardware

6



Focus on: "Typical Disk"



Terms: Platter, Head, Actuator
Cylinder, Track
Sector (physical),
Block (logical), Gap

CS 525

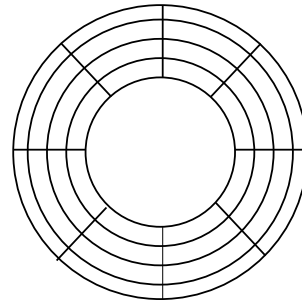


Notes 2 - Hardware

7

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Top View



CS 525



Notes 2 - Hardware

8

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

"Typical" Numbers

Diameter: 1 inch → 15 inches
Cylinders: 100 → 2000
Surfaces: 1 (CDs) →
(Tracks/cyl) 2 (floppies) → 30
Sector Size: 512B → 50K
Capacity: 360 KB (old floppy)
→ 1 TB (I use)

CS 525

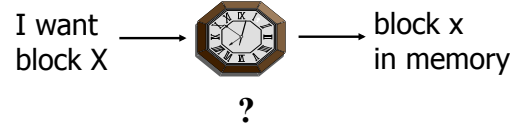


Notes 2 - Hardware

9

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Disk Access Time



CS 525



Notes 2 - Hardware

10

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Time = Seek Time +
Rotational Delay +
Transfer Time +
Other

CS 525

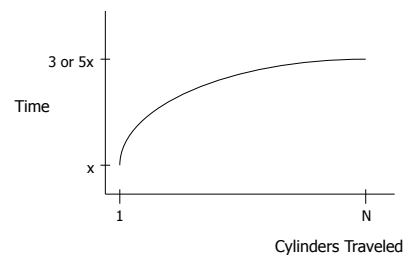


Notes 2 - Hardware

11

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Seek Time



CS 525



Notes 2 - Hardware

12

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Average Random Seek Time

$$S = \frac{\sum_{i=1}^N \sum_{\substack{j=1 \\ j \neq i}}^N \text{SEEKTIME}(i \rightarrow j)}{N(N-1)}$$

CS 525



Notes 2 - Hardware

13

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Average Random Seek Time

$$S = \frac{\sum_{i=1}^N \sum_{\substack{j=1 \\ j \neq i}}^N \text{SEEKTIME}(i \rightarrow j)}{N(N-1)}$$

“Typical” S: 10 ms → 40 ms

CS 525

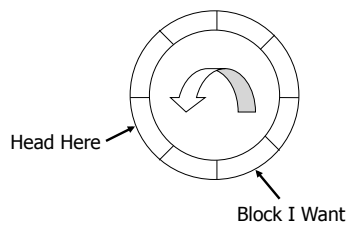


Notes 2 - Hardware

14

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Rotational Delay



CS 525



Notes 2 - Hardware

15

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Average Rotational Delay

$R = 1/2$ revolution

“typical” R = 8.33 ms (3600 RPM)

CS 525



Notes 2 - Hardware

16

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Transfer Rate: t

- “typical” t: 10’s → 100’s MB/second
- transfer time: $\frac{\text{block size}}{t}$

CS 525



Notes 2 - Hardware

17

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Other Delays

- CPU time to issue I/O
- Contention for controller
- Contention for bus, memory

CS 525



Notes 2 - Hardware

18

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Other Delays

- CPU time to issue I/O
- Contention for controller
- Contention for bus, memory

“Typical” Value: 0

CS 525



Notes 2 - Hardware

19

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Other Delays (now and near future)

- Increasing amount of parallelism
- Contention can become a problem
- -> need rethink approach to scale

CS 525



Notes 2 - Hardware

20

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

- So far: Random Block Access
- What about: Reading “Next” block?

CS 525



Notes 2 - Hardware

21

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

If we do things right (e.g., Double Buffer, Stagger Blocks...)

Time to get = $\frac{\text{Block Size}}{t} + \text{Negligible}$
block

- skip gap
- switch track
- once in a while, next cylinder

CS 525



Notes 2 - Hardware

22

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Rule of Thumb Random I/O: Expensive
 Sequential I/O: Much less

- Ex: 1 KB Block
 - » Random I/O: ~ 20 ms.
 - » Sequential I/O: ~ 1 ms.

CS 525



Notes 2 - Hardware

23

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Cost for Writing similar to Reading

... unless we want to verify!
need to add (full) rotation + $\frac{\text{Block size}}{t}$

CS 525



Notes 2 - Hardware

24

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

- To Modify a Block?

CS 525



Notes 2 - Hardware

25

- To Modify a Block?

To Modify Block:

- (a) Read Block
- (b) Modify in Memory
- (c) Write Block
- [(d) Verify?]

CS 525



Notes 2 - Hardware

26

Block Address:

- Physical Device
- Cylinder #
- Surface #
- Sector

CS 525

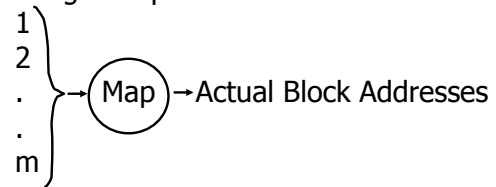


Notes 2 - Hardware

27

Complication: Bad Blocks

- Messy to handle
- May map via software to integer sequence



CS 525



Notes 2 - Hardware

28

An Example Megatron 747 Disk (old)

- 3.5 in diameter
- 3600 RPM
- 1 surface
- 16 MB usable capacity (16×2^{20})
- 128 cylinders
- seek time: average = 25 ms.
adjacent cyl = 5 ms.

CS 525



Notes 2 - Hardware

29

- 1 KB blocks = sectors
- 10% overhead between blocks
- capacity = 16 MB = $(2^{20})16 = 2^{24}$
- # cylinders = $128 = 2^7$
- bytes/cyl = $2^{24}/2^7 = 2^{17} = 128 \text{ KB}$
- blocks/cyl = $128 \text{ KB} / 1 \text{ KB} = 128$

CS 525

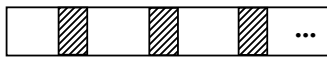


Notes 2 - Hardware

30

3600 RPM → 60 revolutions / sec
 → 1 rev. = 16.66 msec.

One track:



CS 525



Notes 2 - Hardware

31

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

3600 RPM → 60 revolutions / sec
 → 1 rev. = 16.66 msec.

One track:



Time over useful data: $(16.66)(0.9) = 14.99$ ms.
 Time over gaps: $(16.66)(0.1) = 1.66$ ms.
 Transfer time 1 block = $14.99/128 = 0.117$ ms.
 Trans. time 1 block+gap = $16.66/128 = 0.13$ ms.

CS 525



Notes 2 - Hardware

32

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Burst Bandwidth

1 KB in 0.117 ms.

$$BB = 1/0.117 = 8.54 \text{ KB/ms.}$$

or

$$BB = 8.54 \text{ KB/ms} \times 1000 \text{ ms/1sec} \times 1 \text{ MB}/1024 \text{ KB} \\ = 8540/1024 = 8.33 \text{ MB/sec}$$

CS 525



Notes 2 - Hardware

33

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Sustained bandwidth (over track)

128 KB in 16.66 ms.

$$SB = 128/16.66 = 7.68 \text{ KB/ms}$$

or

$$SB = 7.68 \times 1000/1024 = 7.50 \text{ MB/sec.}$$

CS 525



Notes 2 - Hardware

34

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

T_1 = Time to read one random block

$$T_1 = \text{seek} + \text{rotational delay} + TT$$

$$= 25 + (16.66/2) + .117 = 33.45 \text{ ms.}$$

CS 525

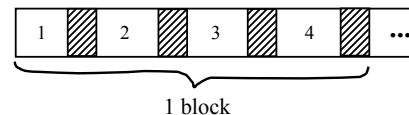


Notes 2 - Hardware

35

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Suppose OS deals with 4 KB blocks



$$T_4 = 25 + (16.66/2) + (.117) \times 1 \\ + (.130) \times 3 = 33.83 \text{ ms}$$

[Compare to $T_1 = 33.45$ ms]

CS 525



Notes 2 - Hardware

36

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

T_T = Time to read a full track
(start at any block)

$$T_T = 25 + (0.130/2) + 16.66^* = 41.73 \text{ ms}$$

↑
to get to first block

* Actually, a bit less; do not have to read last gap.

CS 525



Notes 2 - Hardware

37

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

The NEW Megatron 747

- 8 Surfaces, 3.5 Inch diameter
 - outer 1 inch used
- $2^{13} = 8192$ Tracks/surface
- 256 Sectors/track
- $2^9 = 512$ Bytes/sector

CS 525

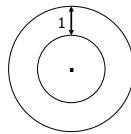


Notes 2 - Hardware

38

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

- 8 GB Disk
- If all tracks have 256 sectors
 - Outermost density: 100,000 bits/inch
 - Inner density: 250,000 bits/inch



CS 525



Notes 2 - Hardware

39

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

- Outer third of tracks: 320 sectors
- Middle third of tracks: 256
- Inner third of tracks: 192
- Density: 114,000 → 182,000 bits/inch

CS 525



Notes 2 - Hardware

40

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Timing for new Megatron 747 (Ex 2.3)

- Time to read 4096-byte block:
 - MIN: 0.5 ms
 - MAX: 33.5 ms
 - AVE: 14.8 ms

CS 525



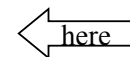
Notes 2 - Hardware

41

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Outline

- Hardware: Disks
- Access Times
- Example: Megatron 747
- Optimizations
- Other Topics
 - Storage Costs
 - Using Secondary Storage
 - Disk Failures



CS 525



Notes 2 - Hardware

42

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Optimizations (in controller or O.S.)

- Disk Scheduling Algorithms
 - e.g., elevator algorithm
- Track (or larger) Buffer
- Pre-fetch
- Arrays
- Mirrored Disks
- On Disk Cache

CS 525



Notes 2 - Hardware

43

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Double Buffering

Problem: Have a File

» Sequence of Blocks B1, B2

Have a Program

» Process B1

» Process B2

» Process B3

⋮

CS 525



Notes 2 - Hardware

44

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Single Buffer Solution

- (1) Read B1 → Buffer
- (2) Process Data in Buffer
- (3) Read B2 → Buffer
- (4) Process Data in Buffer ...

CS 525



Notes 2 - Hardware

45

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Say P = time to process/block
 R = time to read in 1 block
 n = # blocks

Single buffer time = $n(P+R)$

CS 525

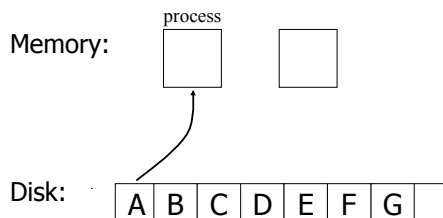


Notes 2 - Hardware

46

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Double Buffering



CS 525

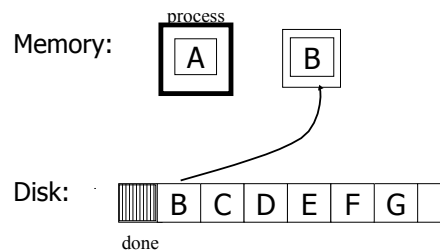


Notes 2 - Hardware

47

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Double Buffering



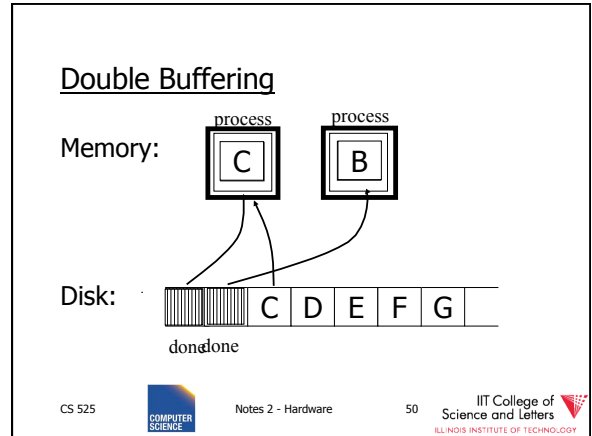
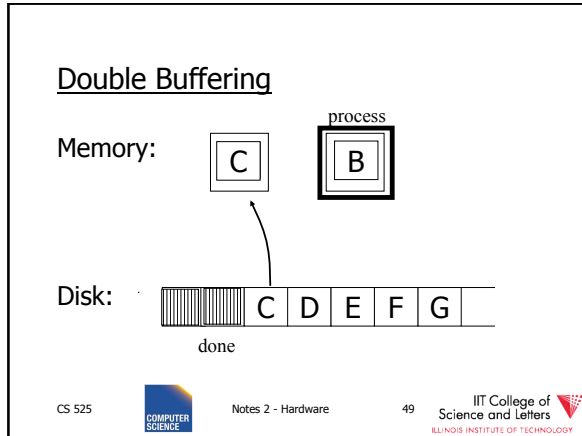
CS 525



Notes 2 - Hardware

48


IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY



Say $P \geq R$

P = Processing time/block
R = IO time/block
n = # blocks

What is processing time?


CS 525  Notes 2 - Hardware 51 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

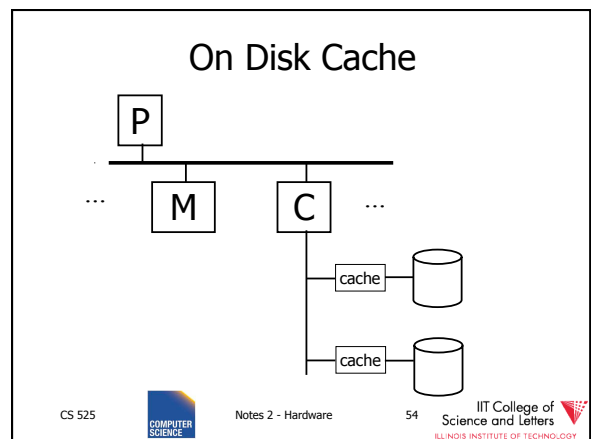
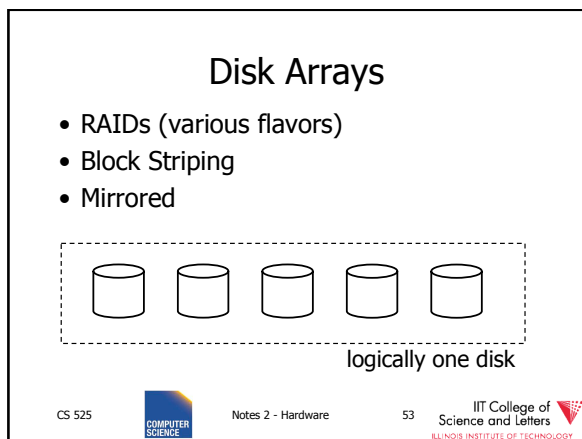
Say $P \geq R$

P = Processing time/block
R = IO time/block
n = # blocks

What is processing time?

- Double buffering time = $R + nP$
- Single buffering time = $n(R+P)$

CS 525  Notes 2 - Hardware 52 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY



Block Size Selection?

- Big Block → Amortize I/O Cost, Less Management Overhead

Unfortunately...

- Big Block ⇒ Read in more useless stuff! and takes longer to read

CS 525



Notes 2 - Hardware

55

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Trend

- As memory prices drop, blocks get bigger ...

CS 525

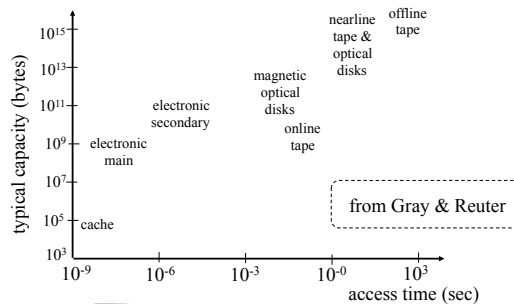


Notes 2 - Hardware

56

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Storage Cost



CS 525

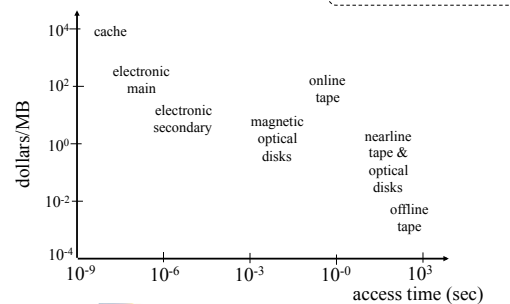


Notes 2 - Hardware

57

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Storage Cost



CS 525



Notes 2 - Hardware

58

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Using secondary storage effectively

- Example: Sorting data on disk
- Conclusion:
 - I/O costs dominate
 - Design algorithms to reduce I/O
- Also: How big should blocks be?

CS 525



Notes 2 - Hardware

59

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Five Minute Rule

- THE 5 MINUTE RULE FOR TRADING MEMORY FOR DISC ACCESSES
Jim Gray & Franco Putzolu
May 1985
- The Five Minute Rule, Ten Years Later
Goetz Graefe & Jim Gray
December 1997

CS 525



Notes 2 - Hardware

60

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Five Minute Rule

- Say a page is accessed every X seconds
- CD = cost if we keep that page on disk
 - \$D = cost of disk unit
 - I = numbers IOs that unit can perform per second
 - In X seconds, unit can do XI IOs
 - So CD = \$D / XI

CS 525



Notes 2 - Hardware

61

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Five Minute Rule

- Say a page is accessed every X seconds
- CM = cost if we keep that page on RAM
 - \$M = cost of 1 MB of RAM
 - P = numbers of pages in 1 MB RAM
 - So CM = \$M / P

CS 525



Notes 2 - Hardware

62

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Five Minute Rule

- Say a page is accessed every X seconds
- If CD is smaller than CM,
 - keep page on disk
 - else keep in memory
- Break even point when CD = CM, or

$$X = \frac{\$D P}{I \$M}$$

CS 525



Notes 2 - Hardware

63

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Using '97 Numbers

- P = 128 pages/MB (8KB pages)
- I = 64 accesses/sec/disk
- \$D = 2000 dollars/disk (9GB + controller)
- \$M = 15 dollars/MB of DRAM
- X = 266 seconds (about 5 minutes)
(did not change much from 85 to 97)

CS 525



Notes 2 - Hardware

64

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Disk Failures

- Partial → Total
- Intermittent → Permanent

CS 525



Notes 2 - Hardware

65

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Coping with Disk Failures

- Detection
 - e.g. Checksum
- Correction
 - ⇒ Redundancy

CS 525



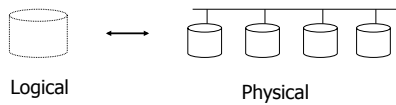
Notes 2 - Hardware

66

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

At what level do we cope?

- Single Disk
 - e.g., Error Correcting Codes
- Disk Array



CS 525

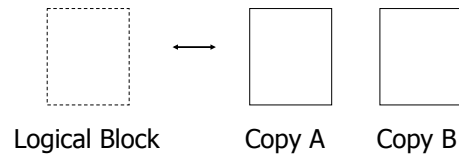


Notes 2 - Hardware

67

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

→ Operating System e.g., Stable Storage



CS 525



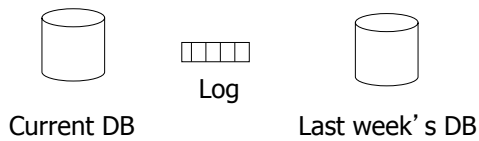
Notes 2 - Hardware

68

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

→ Database System

- e.g.,



CS 525



Notes 2 - Hardware

69

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Summary

- Secondary storage, mainly disks
- I/O times + formulas
 - Sequential vs. random
- I/Os should be avoided,
especially random ones.....
- OS optimizations
- Disk errors

CS 525



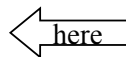
Notes 2 - Hardware

70

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Outline

- Hardware: Disks
- Access Times
- Example: Megatron 747
- Optimizations
- Other Topics
 - Storage Costs
 - Using Secondary Storage
 - Disk Failures



here

CS 525



Notes 2 - Hardware

71

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Outlook - Hardware

- Disk Access is the main limiting factor
- However, to implement fast DBMS
 - need to understand other parts of the hardware
 - Memory hierarchy
 - CPU architecture: pipelining, vector instructions, OOE, ...
 - SSD storage
 - need to understand how OS manages hardware
 - File access, VM, Buffering, ...

CS 525

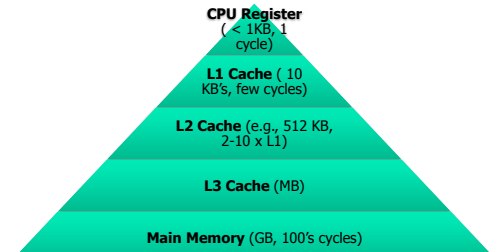


Notes 2 - Hardware

72

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Memory Hierarchy



CS 525



Notes 2 - Hardware

73

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Memory Hierarchy

- **Compare:** Disk vs. Main Memory
- Reduce accesses to main memory
- Cache conscious algorithms

CS 525



Notes 2 - Hardware

74

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Increasing Amount of Parallelism

- Contention on, e.g., Memory
- NUMA
- Algorithmic Challenges
 - How to parallelize algorithms?
 - Sometime: Completely different approach required
 - -> Rewrite large parts of DBMS

CS 525



Notes 2 - Hardware

75

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

New Trend: Software/Hardware Co-design

- Actually, revived trend: database machines (80's)
- New goals: power consumption
- Design specific hardware and write special software for it
- E.g., Oracle Exadata, Oracle Labs

CS 525



Notes 2 - Hardware

76

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

CS 525: Advanced Database Organization

03: Disk Organization



Boris Glavic

Slides: adapted from a [course](#) taught by [Hector Garcia-Molina](#), Stanford InfoLab

CS 525



Notes 3

1

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Topics for today

- How to lay out data on disk
- How to move it to/from memory

CS 525



Notes 3

2

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

What are the data items we want to store?

- a salary
- a name
- a date
- a picture

CS 525



Notes 3

3

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

What are the data items we want to store?

- a salary
- a name
- a date
- a picture

⇒ What we have available: Bytes



CS 525



Notes 3

4

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

To represent:

- Integer (short): 2 bytes
e.g., 35 is

`00000000` `00100011`

Endian! Could as well be

`00100011` `00000000`

- Real, floating point
 n bits for mantissa, m for exponent....

CS 525



Notes 3

5

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

To represent:

- Characters
→ various coding schemes suggested,
most popular is ASCII (1 byte encoding)

Example:

A: 1000001

a: 1100001

5: 0110101

LF: 0001010

CS 525



Notes 3

6

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

To represent:

- Boolean
e.g., TRUE

1111	1111
------	------

FALSE

0000	0000
------	------
- Application specific
e.g., enumeration
RED → 1 GREEN → 3
BLUE → 2 YELLOW → 4 ...

To represent:

- Boolean
e.g., TRUE

1111	1111
------	------

FALSE

0000	0000
------	------
- Application specific
e.g., RED → 1 GREEN → 3
BLUE → 2 YELLOW → 4 ...

⇒ Can we use less than 1 byte/code?

Yes, but only if desperate...

To represent:

- Dates
e.g.: - Integer, # days since Jan 1, 1900
- 8 characters, YYYYMMDD
- 7 characters, YYYYDDD
(not YYMMDD! Why?)
- Time
e.g. - Integer, seconds since midnight
- characters, HHMMSSFF

To represent:

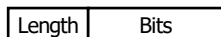
- String of characters
- Null terminated
e.g.,

c	a	t	⊗		
---	---	---	---	--	--
- Length given
e.g.,

3	c	a	t	⊗	
---	---	---	---	---	--
- Fixed length

To represent:

- Bag of bits



Key Point

- Fixed length items
- Variable length items
- usually length given at beginning

Also

- Type of an item: Tells us how to interpret (plus size if fixed)

CS 525



Notes 3

13

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Overview

Data Items



Records



Blocks



Files



Memory

CS 525



Notes 3

14

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Record - Collection of related data items (called FIELDS)

E.g.: Employee record:

name field,
salary field,
date-of-hire field, ...

CS 525



Notes 3

15

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Types of records:

- Main choices:
 - FIXED vs VARIABLE FORMAT
 - FIXED vs VARIABLE LENGTH

CS 525



Notes 3

16

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Fixed format

A SCHEMA (not record) contains following information

- # fields
- type of each field
- order in record
- meaning of each field

CS 525



Notes 3

17

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Example: fixed format and length

Employee record

- (1) E#, 2 byte integer
- (2) E.name, 10 char.
- (3) Dept, 2 byte code

} Schema

55 | s m i t h | 02

83 | j o n e s | 01

} Records

CS 525



Notes 3

18

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Variable format

- Record itself contains format
“Self Describing”

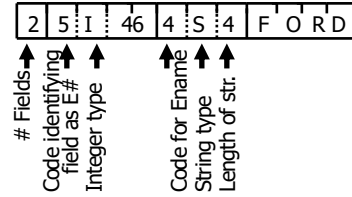
CS 525



Notes 3

19

Example: variable format and length



Field name codes could also be strings, i.e. TAGS

CS 525



Notes 3

20

Variable format useful for:

- “sparse” records
- repeating fields
- evolving formats

...► But may waste space...
Additional indirection...

CS 525



Notes 3

21

- EXAMPLE:** var format record with repeating fields
Employee → one or more → children

3	E_name: Fred	Child: Sally	Child: Tom
---	--------------	--------------	------------

CS 525



Notes 3

22

Note: Repeating fields does not imply
- variable format, nor
- variable size

John	Sailing	Chess	--
------	---------	-------	----

CS 525



Notes 3

23

Note: Repeating fields does not imply
- variable format, nor
- variable size

John	Sailing	Chess	--
------	---------	-------	----

- Key is to allocate maximum number of repeating fields (if not used → null)

CS 525

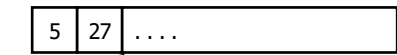


Notes 3

24

☆ Many variants between fixed - variable format:

Example: Include record type in record



record type tells me what to expect (i.e. points to schema)

record length

Record header - data at beginning that describes record

May contain:

- record type
- record length
- time stamp
- null-value bitmap
- other stuff ...

Other interesting issues:

- Compression
 - within record - e.g. code selection
 - collection of records - e.g. find common patterns
- Encryption
- Splitting of large records
 - E.g., image field, store pointer

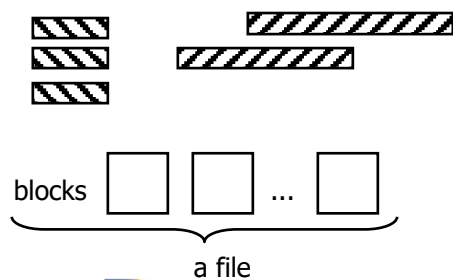
Record Header – null-map

- SQL: NULL is special value for every data type
 - Reserve one value for each data type as NULL?
- Easier solution
 - Record header has a bitmap to store whether field is NULL
 - Only store non-NULL fields in record

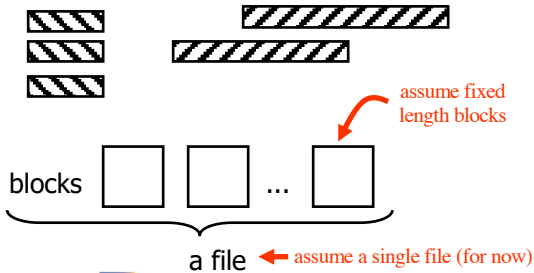
Separate Storage of Large Values

- Store fields with large values separately
 - E.g., image or binary document
 - Records have pointers to large field content
- Rationale
 - Large fields mostly not used in search conditions
 - Benefit from smaller records

Next: placing records into blocks



Next: placing records into blocks



CS 525



Notes 3

31

Options for storing records in blocks:

- (1) separating records
- (2) spanned vs. unspanned
- (3) sequencing
- (4) indirection

CS 525



Notes 3

32

(1) Separating records



- (a) no need to separate - fixed size recs.
- (b) special marker
- (c) give record lengths (or offsets)
 - within each record
 - in block header

CS 525



Notes 3

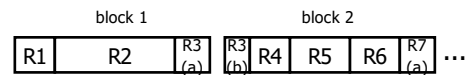
33

(2) Spanned vs. Unspanned

- Unspanned: records must be within one block



- Spanned



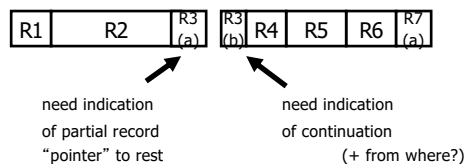
CS 525



Notes 3

34

With spanned records:



CS 525



Notes 3

35

Spanned vs. unspanned:

- Unspanned is much simpler, but may waste space...
- Spanned essential if record size > block size

CS 525



Notes 3

36

(3) Sequencing

- Ordering records in file (and block) by some key value

Sequential file (\Rightarrow sequenced)

CS 525



Notes 3

37

Why sequencing?

Typically to make it possible to efficiently read records in order
(e.g., to do a merge-join — discussed later)

CS 525

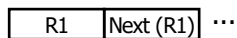


Notes 3

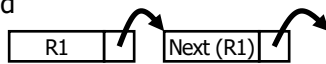
38

Sequencing Options

(a) Next record physically contiguous



(b) Linked



CS 525



Notes 3

39

Sequencing Options

(c) Overflow area

Records in sequence

R1
R2
R3
R4
R5

CS 525



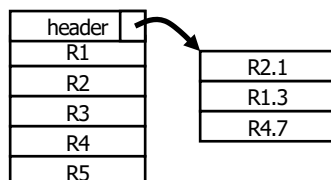
Notes 3

40

Sequencing Options

(c) Overflow area

Records in sequence



CS 525

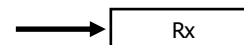


Notes 3

41

(4) Indirection

- How does one refer to records?



CS 525



Notes 3

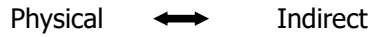
42

(4) Indirection

- How does one refer to records?



Many options:



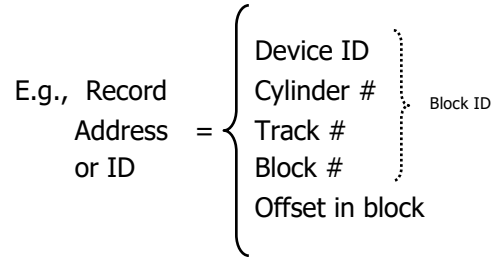
CS 525



Notes 3

43

☆ Purely Physical



CS 525

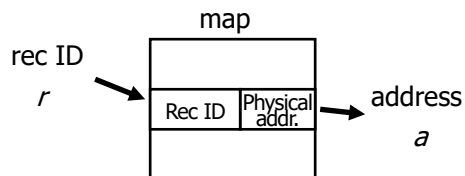


Notes 3

44

☆ Fully Indirect

E.g., Record ID is arbitrary bit string



CS 525



Notes 3

45

Tradeoff

Flexibility \longleftrightarrow Cost
to move records of indirection
(for deletions, insertions)

CS 525



Notes 3

46

Physical \longleftrightarrow Indirect

↑
Many options
in between ...

CS 525



Notes 3

47

Block header - data at beginning that describes block

May contain:

- File ID (or RELATION or DB ID)
- This block ID
- Record directory
- Pointer to free space
- Type of block (e.g. contains recs type 4; is overflow, ...)
- Pointer to other blocks "like it"
- Timestamp ...

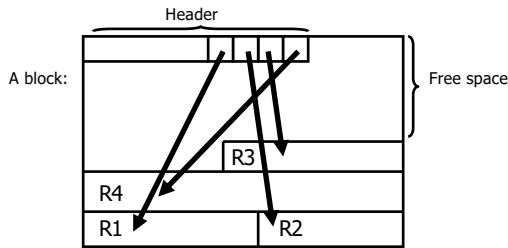
CS 525



Notes 3

48

Example: Indirection in block



CS 525



Notes 3

49

Tuple Identifier (TID)

- TID is
 - Page identifier
 - Slot number
- Slot stores either record or pointer (TID)
- TID of a record is fixed for all time

CS 525



Notes 3

50

TID Operations

- Insertion
 - Set TID to record location (page, slot)
- Moving record
 - e.g., update variable-size or reorganization
 - Case 1: TID points to record
 - Replace record with pointer (new TID)
 - Case 2: TID points to pointer (TID)
 - Replace pointer with new pointer

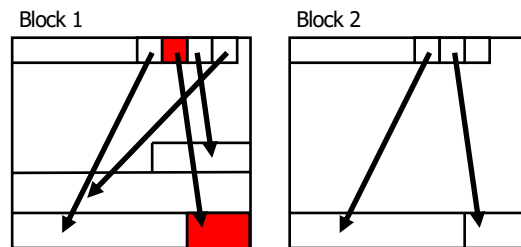
CS 525



Notes 3

51

TID: Block 1, Slot 2



CS 525

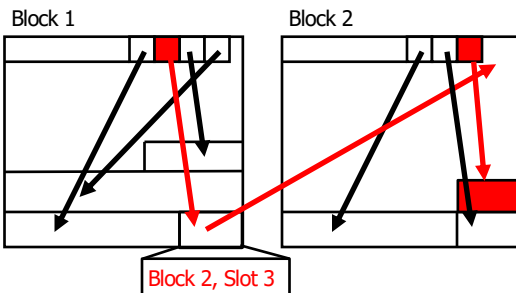


Notes 3

52

Move record to Block 2 slot 3 -> TID does not change!

TID: Block 1, Slot 2



CS 525

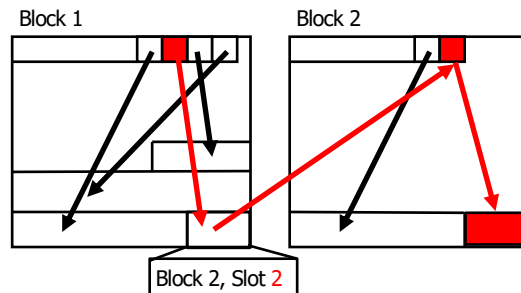


Notes 3

53

Move record again to Block 2 slot 2 -> still one level of indirection

TID: Block 1, Slot 2



CS 525



Notes 3

54

TID Properties

- TID of record never changes
 - Can be used safely as pointer to record (e.g., in index)
- At most one level of indirection
 - Relatively efficient
 - Changes to physical address - changing max 2 pages

CS 525



Notes 3

55

Options for storing records in blocks:

- (1) separating records
- (2) spanned vs. unspanned
- (3) sequencing
- (4) indirection

CS 525



Notes 3

56

Other Topics

- (1) Insertion/Deletion
- (2) Buffer Management
- (3) Comparison of Schemes

CS 525

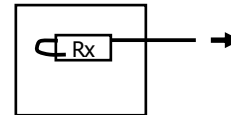


Notes 3

57

Deletion

Block



CS 525



Notes 3

58

Options:

- (a) Immediately reclaim space
- (b) Mark deleted

CS 525



Notes 3

59

Options:

- (a) Immediately reclaim space
- (b) Mark deleted
 - May need chain of deleted records (for re-use)
 - Need a way to mark:
 - special characters
 - delete field
 - in map

CS 525



Notes 3

60

☆ As usual, many tradeoffs...

- How expensive is it to move valid record to free space for immediate reclaim?
- How much space is wasted?
 - e.g., deleted records, delete fields, free space chains,...

CS 525

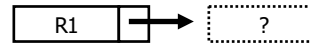


Notes 3

61

Concern with deletions

Dangling pointers



CS 525



Notes 3

62

Solution #1: Do not worry

CS 525



Notes 3

63

Solution #2: Tombstones

E.g., Leave “MARK” in map or old location

CS 525



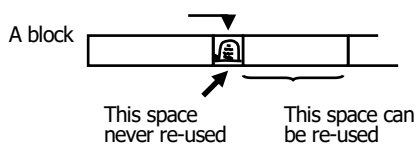
Notes 3

64

Solution #2: Tombstones

E.g., Leave “MARK” in map or old location

- Physical IDs



CS 525



Notes 3

65

Solution #2: Tombstones

E.g., Leave “MARK” in map or old location

- Logical IDs

map	
ID	LOC
7788	

Never reuse ID 7788 nor space in map...

CS 525



Notes 3

66

Insert

Easy case: records not in sequence

- Insert new record at end of file or in deleted slot
- If records are variable size, not as easy...

CS 525



Notes 3

67

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Insert

Hard case: records in sequence

- If free space “close by”, not too bad...
- Or use overflow idea...

CS 525



Notes 3

68

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Interesting problems:

- How much free space to leave in each block, track, cylinder?
- How often do I reorganize file + overflow?

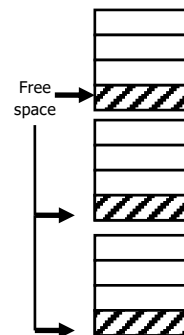
CS 525



Notes 3

69

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY



CS 525



Notes 3

70

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Buffer Management

- For Caching of Disk Blocks
- Buffer Replacement Strategies
 - E.g., LRU, clock
- Pinned blocks
- Forced output — — —▶ in Notes02
- Double buffering
- Swizzling

CS 525



Notes 3

71

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Buffer Manager

- Manages blocks cached from disk in main memory
- Usually -> fixed size buffer (M pages)
- DB requests page from Buffer Manager
 - Case 1: page is in memory -> return address
 - Case 2: page is on disk -> load into memory, return address

CS 525



Notes 3

72

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Goals

- Reduce the amount of I/O
- Maximize the *hit rate*
 - Ratio of number of page accesses that are fulfilled without reading from disk
- -> Need strategy to decide when to

CS 525



Notes 3

73

Buffer Manager Organization

- Bookkeeping
 - Need to map (hash table) page-ids to locations in buffer (**page frames**)
 - Per page store *fix count, dirty bit, ...*
 - Manage free space
- Replacement strategy
 - If page is requested but buffer is full
 - Which page to emit remove from buffer

CS 525



Notes 3

74

FIFO

- **First In, First Out**
- Replace page that has been in the buffer for the longest time
- Implementation: E.g., pointer to oldest page (circular buffer)
 - Pointer->next = Pointer++ % M
- Simple, but not prioritizing frequently accessed pages

CS 525



Notes 3

75

LRU

- Least Recently Used
- Replace page that has not been accessed for the longest time
- Implementation:
 - List, ordered by LRU
 - Access a page, move it to list tail
- Widely applied and reasonable performance

CS 525



Notes 3

76

Clock

- Frames are organized clock-wise
- Pointer *S* to current frame
- Each frame has a reference bit
 - Page is loaded or accessed -> bit = 1
- Find page to replace (advance pointer)
 - Return first frame with bit = 0
 - On the way set all bits to 0

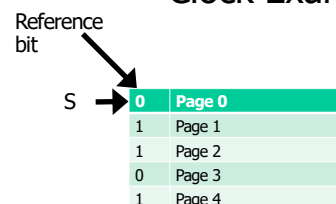
CS 525



Notes 3

77

Clock Example



CS 525



Notes 3

78

Other Replacement Strategies

- LRU-K
- GCLOCK
- Clock-Pro
- ARC
- LFU

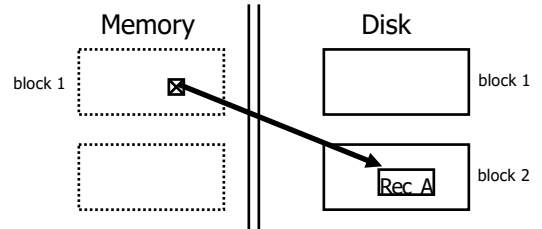
CS 525



Notes 3

79

Swizzling



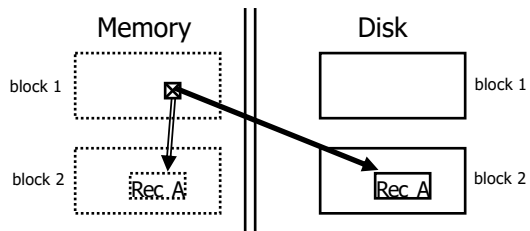
CS 525



Notes 3

80

Swizzling



CS 525



Notes 3

81

Row vs Column Store

- So far we assumed that fields of a record are stored contiguously (row store)...
- Another option is to store all values of a field together (column store)

CS 525



Notes 3

82

Row Store

- Example: Order consists of
 - id, cust, prod, store, price, date, qty

id1	cust1	prod1	store1	price1	date1	qty1
id2	cust2	prod2	store2	price2	date2	qty2
id3	cust3	prod3	store3	price3	date3	qty3

CS 525



Notes 3

83

Column Store

- Example: Order consists of
 - id, cust, prod, store, price, date, qty

id1	cust1	id1	prod1	id1	price1	qty1
id2	cust2	id2	prod2	id2	price2	qty2
id3	cust3	id3	prod3	id3	price3	qty3
id4	cust4	id4	prod4	id4	price4	qty4
...

ids may or may not be stored explicitly

CS 525



Notes 3

84

Row vs Column Store

- Advantages of Column Store
 - more compact storage (fields need not start at byte boundaries)
 - Efficient compression, e.g., RLE
 - efficient reads on data mining operations
- Advantages of Row Store
 - writes (multiple fields of one record) more efficient
 - efficient reads for record access (OLTP)

CS 525



Notes 3

85

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Compression

- When should I compress
 - **Compression reduces storage size**
 - Less space on disk
 - More "content" can be read/written with less I/O
 - **(De-)Compression takes time**
 - CPU occupied with compressing/decompressing data -> not available for other operations

CS 525



Notes 3

86

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

The Laws of Compression :-)

- If I/O is the performance bottleneck then compression improves performance
- If CPU is the bottleneck then compression may hurt performance

CS 525



Notes 3

87

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Types of compression

- Dictionary compression
- Run-length encoding (more later)
- Deltacoding (more later)
- Bitpacking
- ...

CS 525



Notes 3

88

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Scope of compression

- Global
 - Global dictionary encoding for strings
 - Replace individual strings with integers using a invertible map
- Per table / column
 - Run-length encode the values of a column
- Per page (group of pages)
 - Compress pages before writing to disk

CS 525



Notes 3

89

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Processing compressed data

- Can we evaluate operations directly over compressed data?
- In some cases yes
- Example: dictionary compressed strings

- **WHERE** name = 'Peter'
- => **WHERE** name = 1

String	Code
Peter	1
Bob	2
Alice	3

CS 525



Notes 3

90

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Example: Apache Parquet

- Parquet is a columnar/compressed storage format developed in the context of the Hadoop ecosystem
- Supported by many big data systems like Spark or MR
- Support nested relational data (we ignore this here)

CS 525



Notes 3

91

Parquet - Structure

- **Row group:** A logical horizontal partitioning of the data into rows
- **Column chunk:** A chunk of the data for a particular column.
 - Guaranteed to be contiguous in the file
- **Page:** Column chunks are divided up into pages, indivisible units for compression and coding

CS 525



Notes 3

92

Parquet - Structure

- **Row group:** GBs in size
- **Column chunk:** typically 100s of MBs
- **Page:** recommended 8KB
 - Pages are compressed and maybe RLE

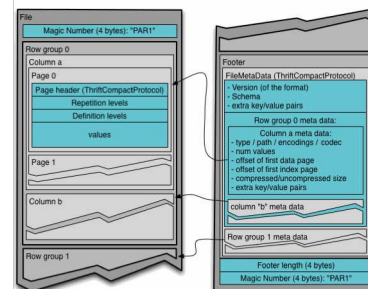
CS 525



Notes 3

93

Parquet - Structure



CS 525



Notes 3

94

Parquet - Analysis

- Columnar
- Hierarchical organization
- Metadata separable from data
- I/O granularity (chunks) different from compression/lookup granularity (pages)

CS 525



Notes 3

95

Comparison

- There are 10,000,000 ways to organize my data on disk...

Which is right for me?

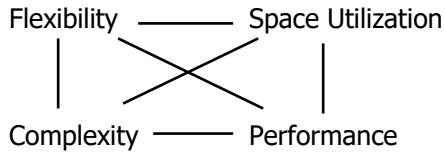
CS 525



Notes 3

96

Issues:



CS 525



Notes 3

97

★ To evaluate a given strategy, compute following parameters:

-> space used for expected data

-> expected time to

- fetch record given key
- fetch record with next key
- insert record
- append record
- delete record
- update record
- read complete file
- reorganize file

CS 525



Notes 3

98

Example

How would you design Megatron 3000 storage system? (for a relational DB, low end)

- Variable length records?
- Spanned?
- What data types?
- Fixed format?
- Record IDs ?
- Sequencing?
- How to handle deletions?

CS 525

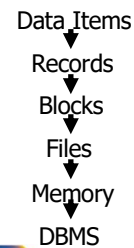


Notes 3

99

Summary

- How to lay out data on disk



CS 525



Notes 3

100

Next

How to find a record quickly,
given a key

CS 525



Notes 3

101

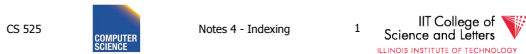
CS 525: Advanced Database Organization

04: Indexing



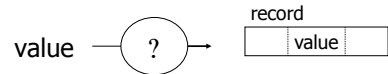
Boris Glavic

Slides: adapted from a [course](#) taught by [Hector Garcia-Molina](#), Stanford InfoLab



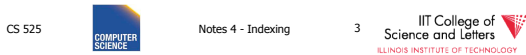
Part 04

Indexing & Hashing



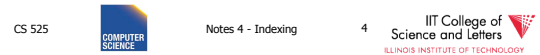
Query Types:

- **Point queries:**
 - *Input:* value **v** of attribute **A**
 - *Output:* all objects (tuples) with that value in attribute **A**
- **Range queries:**
 - *Input:* value interval [**low,high**] of attr **A**
 - *Output:* all tuples with a value **low ≤ v < high** in attribute **A**



Index Considerations:

- Supported Query Types
- Secondary-storage capable
- Storage size
 - Index Size / Data Size
- Complexity of Operations
 - E.g., insert is $O(\log(n))$ worst-case
- Efficient Concurrent Operations?

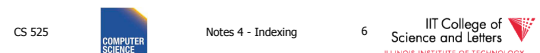
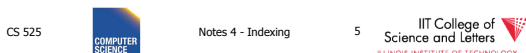


Topics

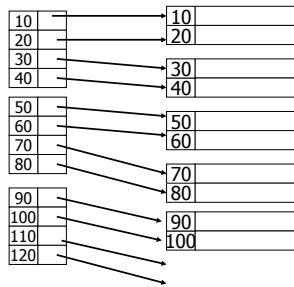
- Conventional indexes
- B-trees
- Hashing schemes
- Advanced Index Techniques

Sequential File

10	
20	
30	
40	
50	
60	
70	
80	
90	
100	



Dense Index Sequential File



CS 525

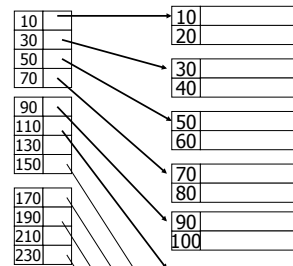


Notes 4 - Indexing

7

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Sparse Index Sequential File



CS 525

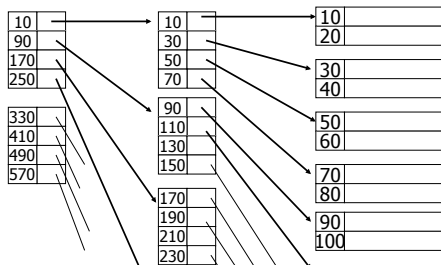


Notes 4 - Indexing

8

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Sparse 2nd level Sequential File



CS 525



Notes 4 - Indexing

9

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

• Comment:

{FILE,INDEX} may be contiguous or not (blocks chained)

CS 525



Notes 4 - Indexing

10

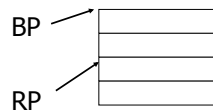
IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Question:

- Can we build a dense, 2nd level index for a dense index?

Notes on pointers:

- (1) Block pointer (sparse index) can be smaller than record pointer



CS 525



Notes 4 - Indexing

11

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

CS 525



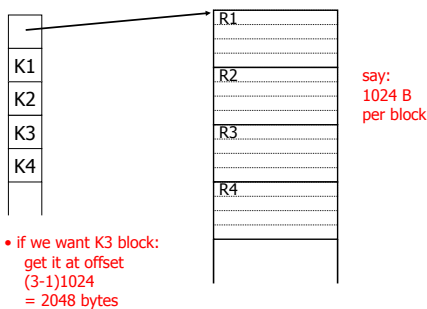
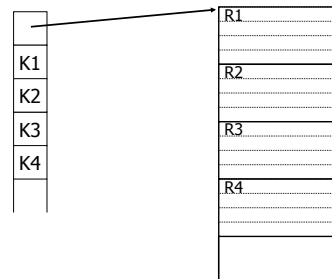
Notes 4 - Indexing

12

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Notes on pointers:

(2) If file is contiguous, then we can omit pointers (i.e., compute them)



Sparse vs. Dense Tradeoff

- **Sparse:** Less index space per record can keep more of index in memory
- **Dense:** Can tell if any record exists without accessing file

(Later:

- sparse better for insertions
- dense needed for secondary indexes)

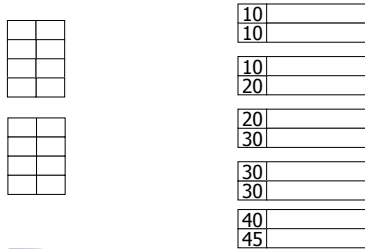
Terms

- Index sequential file
- Search key (≠ primary key)
- Primary index (on Sequencing field)
- Secondary index
- Dense index (all Search Key values in)
- Sparse index
- Multi-level index

Next:

- Duplicate keys
- Deletion/Insertion
- Secondary indexes

Duplicate keys



CS 525



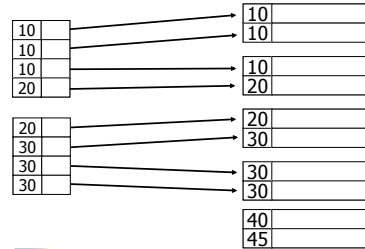
Notes 4 - Indexing

19

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Duplicate keys

Dense index, one way to implement?



CS 525



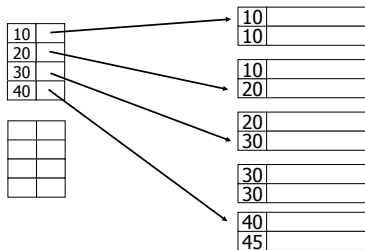
Notes 4 - Indexing

20

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Duplicate keys

Dense index, better way?



CS 525



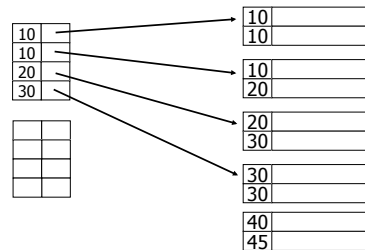
Notes 4 - Indexing

21

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Duplicate keys

Sparse index, one way?



CS 525



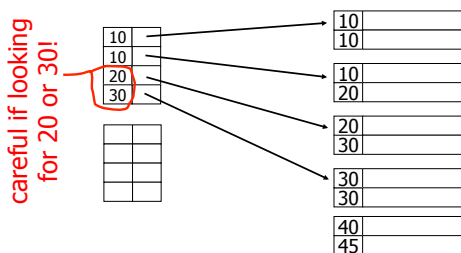
Notes 4 - Indexing

22

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Duplicate keys

Sparse index, one way?



careful if looking for 20 or 30!

CS 525



Notes 4 - Indexing

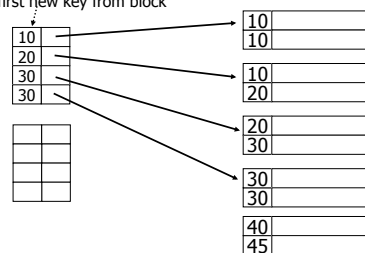
23

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Duplicate keys

Sparse index, another way?

- place first new key from block



CS 525



Notes 4 - Indexing

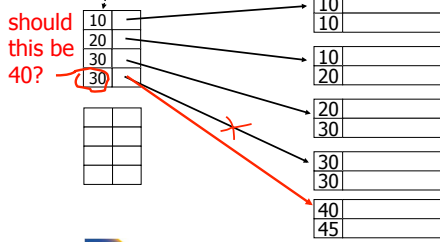
24

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Duplicate keys

Sparse index, another way?

– place first new key from block



CS 525



Notes 4 - Indexing

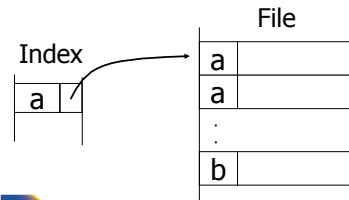
25

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Summary

Duplicate values, primary index

- Index may point to first instance of each value only



CS 525

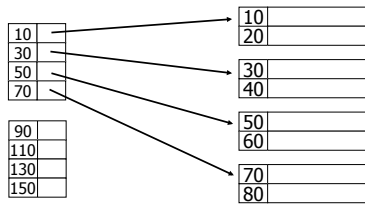


Notes 4 - Indexing

26

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Deletion from sparse index



CS 525



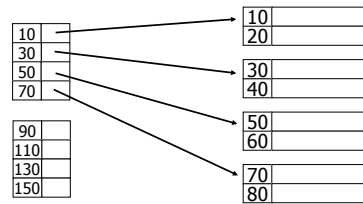
Notes 4 - Indexing

27

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Deletion from sparse index

– delete record 40



CS 525



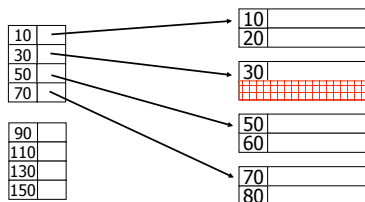
Notes 4 - Indexing

28

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Deletion from sparse index

– delete record 40



CS 525



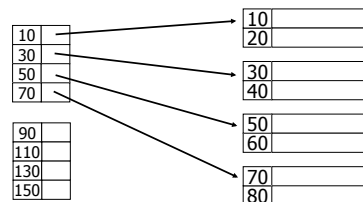
Notes 4 - Indexing

29

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Deletion from sparse index

– delete record 30



CS 525



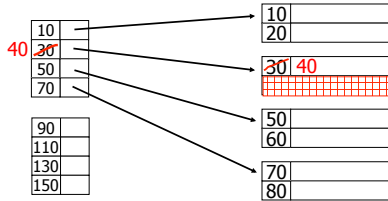
Notes 4 - Indexing

30

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Deletion from sparse index

- delete record 30



CS 525



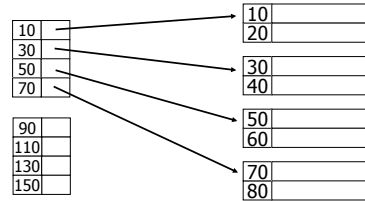
Notes 4 - Indexing

31

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Deletion from sparse index

- delete records 30 & 40



CS 525



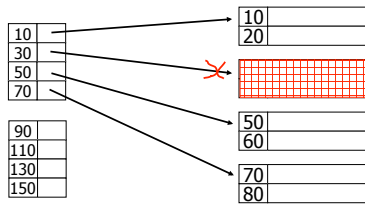
Notes 4 - Indexing

32

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Deletion from sparse index

- delete records 30 & 40



CS 525



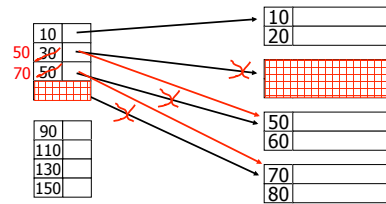
Notes 4 - Indexing

33

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Deletion from sparse index

- delete records 30 & 40



CS 525

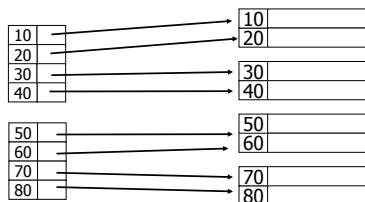


Notes 4 - Indexing

34

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Deletion from dense index



CS 525



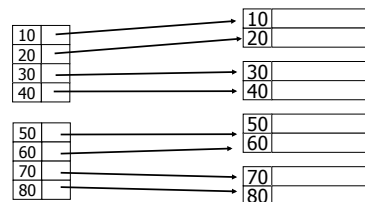
Notes 4 - Indexing

35

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Deletion from dense index

- delete record 30



CS 525



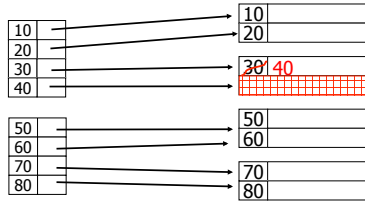
Notes 4 - Indexing

36

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

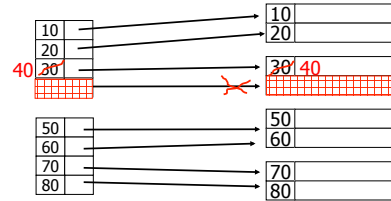
Deletion from dense index

- delete record 30

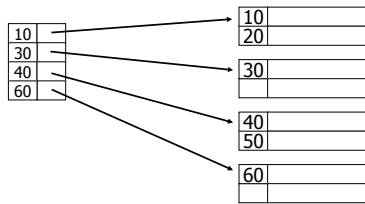


Deletion from dense index

- delete record 30

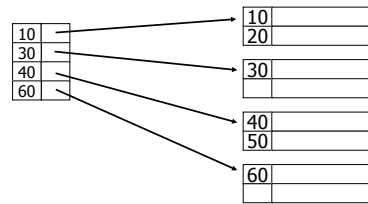


Insertion, sparse index case



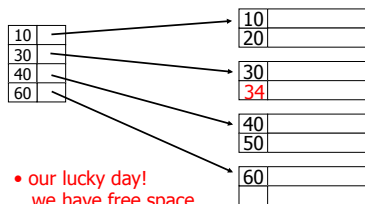
Insertion, sparse index case

- insert record 34



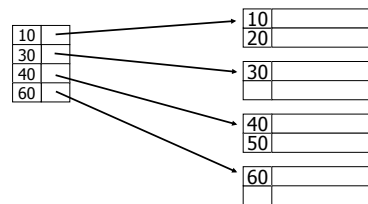
Insertion, sparse index case

- insert record 34



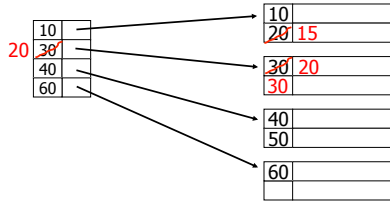
Insertion, sparse index case

- insert record 15



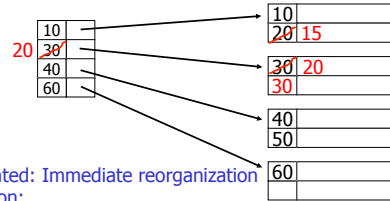
Insertion, sparse index case

- insert record 15



Insertion, sparse index case

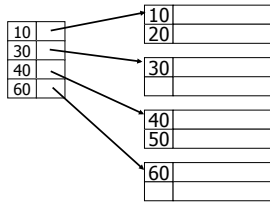
- insert record 15



- Illustrated: Immediate reorganization
- Variation:
 - insert new block (chained file)
 - update index

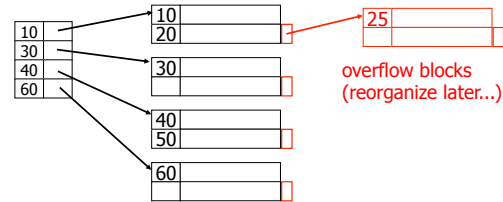
Insertion, sparse index case

- insert record 25



Insertion, sparse index case

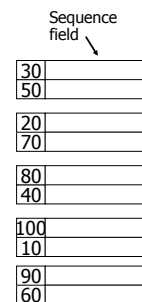
- insert record 25



Insertion, dense index case

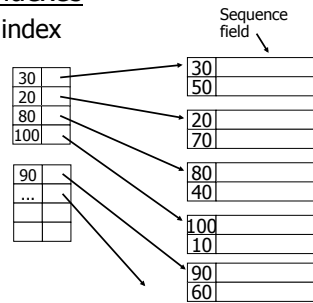
- Similar
- Often more expensive . . .

Secondary indexes



Secondary indexes

- Sparse index



CS 525



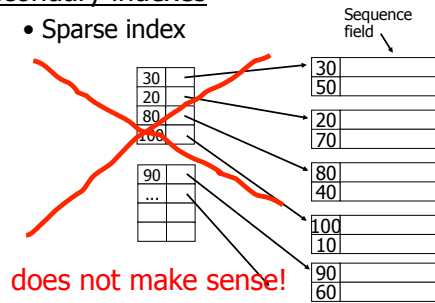
Notes 4 - Indexing

49

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Secondary indexes

- Sparse index



CS 525



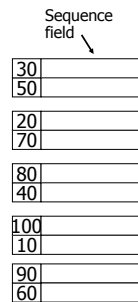
Notes 4 - Indexing

50

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Secondary indexes

- Dense index



CS 525



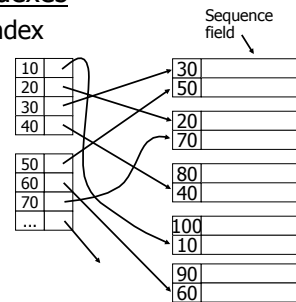
Notes 4 - Indexing

51

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Secondary indexes

- Dense index



CS 525



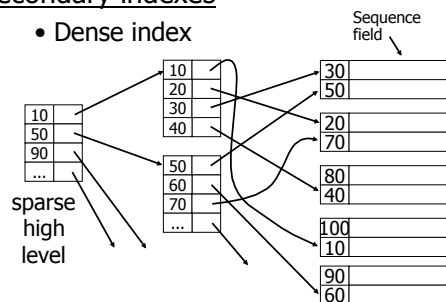
Notes 4 - Indexing

52

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Secondary indexes

- Dense index



CS 525



Notes 4 - Indexing

53

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

With secondary indexes:

- Lowest level is dense
- Other levels are sparse

Also: Pointers are record pointers
(not block pointers; not computed)

CS 525

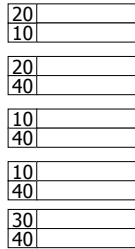


Notes 4 - Indexing

54

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Duplicate values & secondary indexes



CS 525



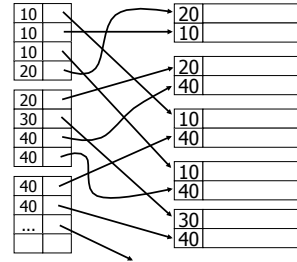
Notes 4 - Indexing

55

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Duplicate values & secondary indexes

one option...



CS 525



Notes 4 - Indexing

56

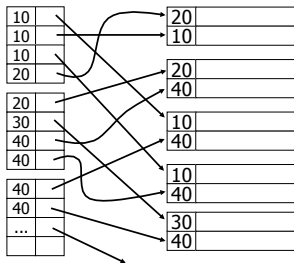
IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Duplicate values & secondary indexes

one option...

Problem:
excess overhead!

- disk space
- search time



CS 525



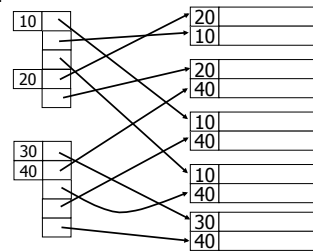
Notes 4 - Indexing

57

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Duplicate values & secondary indexes

another option...



CS 525



Notes 4 - Indexing

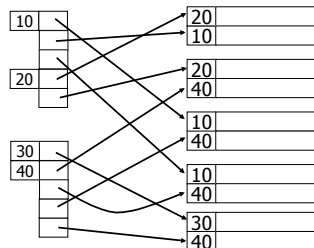
58

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Duplicate values & secondary indexes

another option...

Problem:
variable size records in index!



CS 525

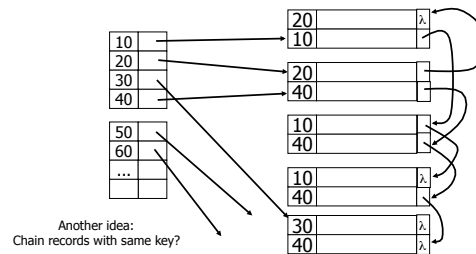


Notes 4 - Indexing

59

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Duplicate values & secondary indexes



CS 525

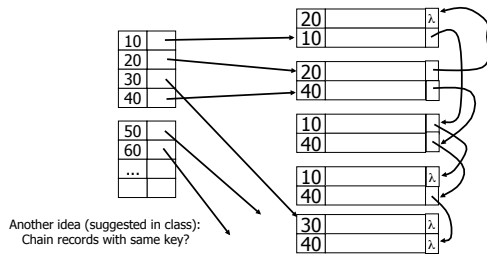


Notes 4 - Indexing

60

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Duplicate values & secondary indexes



Another idea (suggested in class):
Chain records with same key?

- Problems:
- Need to add fields to records
 - Need to follow chain to know records

CS 525

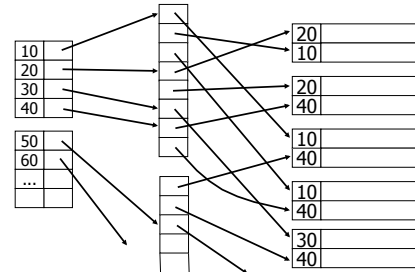


Notes 4 - Indexing

61

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Duplicate values & secondary indexes



buckets

CS 525



Notes 4 - Indexing

62

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Why "bucket" idea is useful

Indexes

Name: primary

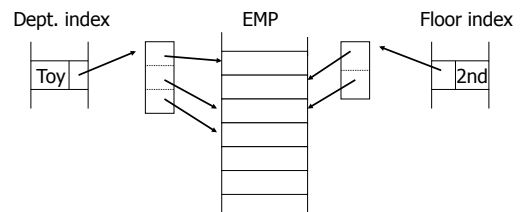
Dept: secondary

Floor: secondary

Records

EMP (name,dept,floor,...)

Query: Get employees in
(Toy Dept) \wedge (2nd floor)



CS 525



Notes 4 - Indexing

63

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

CS 525

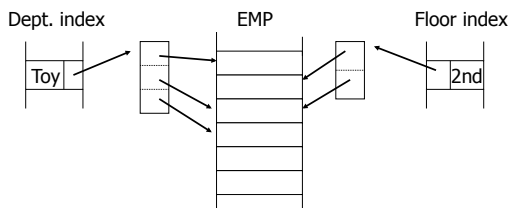


Notes 4 - Indexing

64

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Query: Get employees in
(Toy Dept) \wedge (2nd floor)



→ Intersect toy bucket and 2nd Floor
bucket to get set of matching EMP's

CS 525



Notes 4 - Indexing

65

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

This idea used in
text information retrieval

Documents

...the cat is
fat ...

...was raining
cats and dogs...

...Fido the
dog ...

CS 525

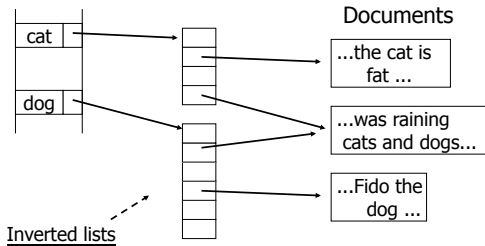


Notes 4 - Indexing

66

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

This idea used in
text information retrieval



IR QUERIES

- Find articles with “cat” and “dog”
- Find articles with “cat” or “dog”
- Find articles with “cat” and not “dog”

Summary so far

- Conventional index
 - Basic Ideas: sparse, dense, multi-level...
 - Duplicate Keys
 - Deletion/Insertion
 - Secondary indexes
 - Buckets of Postings List

Conventional indexes

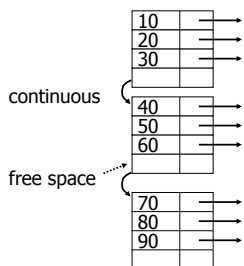
Advantage:

- Simple
- Index is sequential file good for scans

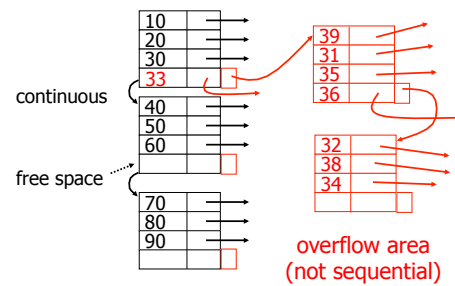
Disadvantage:

- Inserts expensive, and/or
- Lose sequentiality & balance

Example Index (sequential)



Example Index (sequential)



Outline:

- Conventional indexes
- B-Trees ⇒ NEXT
- Hashing schemes
- Advanced Index Techniques

- NEXT: Another type of index
 - Give up on sequentiality of index
 - Try to get “balance”

B+-tree Motivation

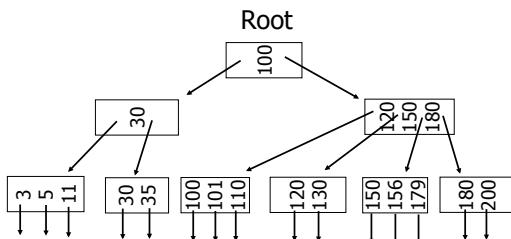
- Tree indices are pretty efficient
 - E.g., binary search tree
 - Average case $O(\log(n))$ lookup
- However
 - Unclear how to map to disk (index larger than main memory, loading partial index)
 - Worst-case $O(n)$ lookup

B+-tree Properties

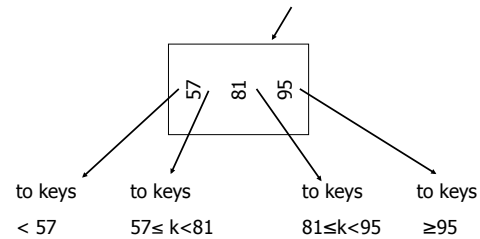
- Large nodes:
 - Node size is multiple of block size
 - -> small number of levels
 - -> simple way to map index to disk
 - -> many keys per node
- Balance:
 - Require all nodes to be more than X% full
 - -> for n records guaranteed only logarithmically many levels
 - -> $\log(n)$ worst-case performance

B+Tree Example

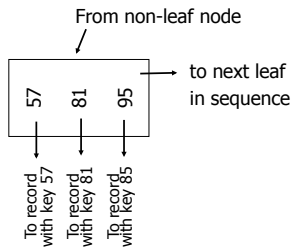
n=3



Sample non-leaf



Sample leaf node:



CS 525



Notes 4 - Indexing

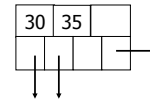
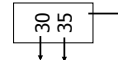
79



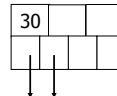
In textbook's notation

$n=3$

Leaf:



Non-leaf:



CS 525



Notes 4 - Indexing

80



Size of nodes: $\begin{cases} n+1 \text{ pointers} \\ n \text{ keys} \end{cases}$ (fixed)

Don't want nodes to be too empty

- Use at least (balance)

Non-leaf: $\lfloor (n+1)/2 \rfloor$ pointers

Leaf: $\lfloor (n+1)/2 \rfloor$ pointers to data

CS 525



Notes 4 - Indexing

81



CS 525

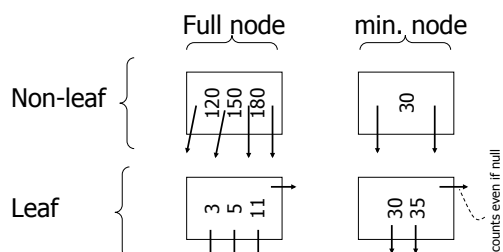


Notes 4 - Indexing

82



$n=3$



CS 525



Notes 4 - Indexing

83



CS 525



Notes 4 - Indexing

84



B+tree rules tree of order n

- (1) All leaves at same lowest level (balanced tree)
 - > guaranteed worst-case complexity for operations on the index
- (2) Pointers in leaves point to records except for "sequence pointer"

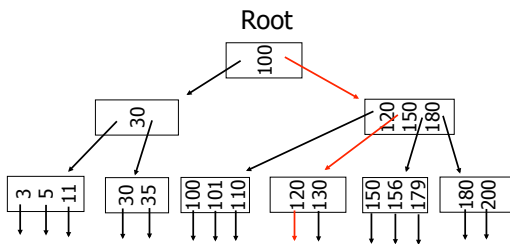
(3) Number of pointers/keys for B+tree

	Max ptrs	Max keys	Min ptrs-data	Min keys
Non-leaf (non-root)	$n+1$	n	$\lceil (n+1)/2 \rceil$	$\lceil (n+1)/2 \rceil - 1$
Leaf (non-root)	$n+1$	n	$\lfloor (n+1)/2 \rfloor$	$\lfloor (n+1)/2 \rfloor$
Root	$n+1$	n	1	1

Search Algorithm

- Search for key **k**
- Start from root until leaf is reached
- For current node find i so that
 - $\text{Key}[i] \leq k < \text{Key}[i + 1]$
 - Follow $i+1^{\text{th}}$ pointer
- If current node is leaf return pointer to record or fail (no such record in tree)

Search Example $k=120$ $n=3$



Remarks Search

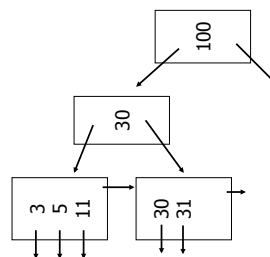
- If n is large, e.g., 500
- Keys inside node are sorted
- \rightarrow use binary search to find i
- Performance considerations
 - Linear search $O(n)$
 - Binary search $O(\log_2(n))$

Insert into B+tree

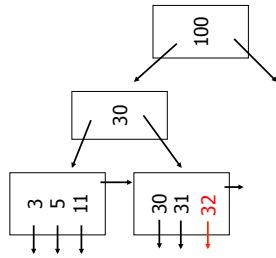
- simple case
 - space available in leaf
- leaf overflow
- non-leaf overflow
- new root

(a) Insert key = 32

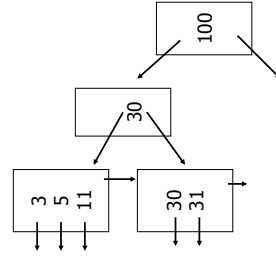
$n=3$



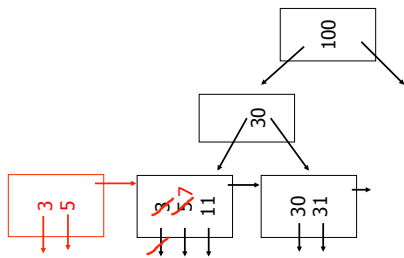
(a) Insert key = 32 n=3



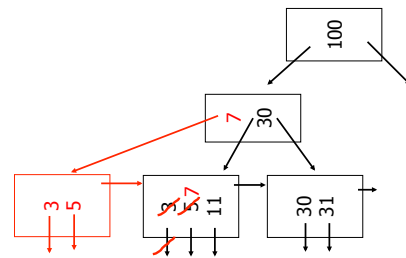
(a) Insert key = 7 n=3



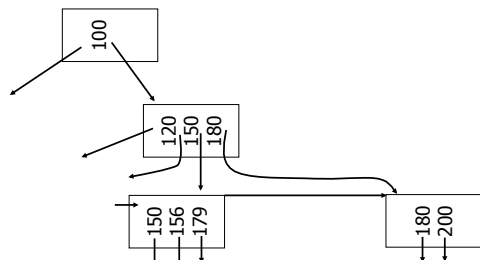
(a) Insert key = 7 n=3



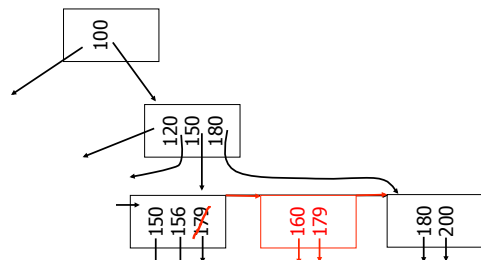
(a) Insert key = 7 n=3



(c) Insert key = 160 n=3

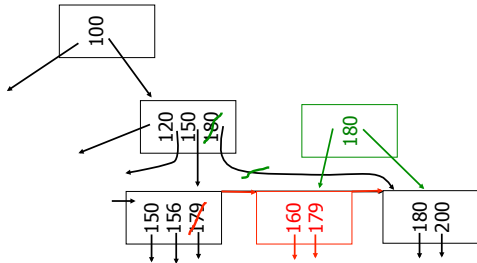


(c) Insert key = 160 n=3



(c) Insert key = 160

n=3



CS 525



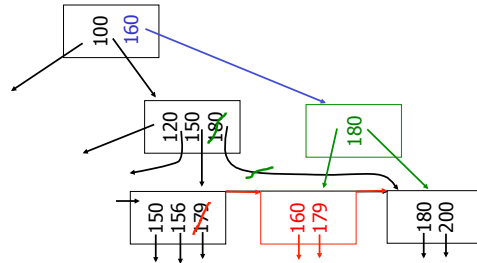
Notes 4 - Indexing

97

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

(c) Insert key = 160

n=3



CS 525



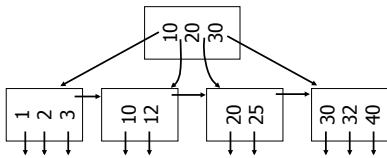
Notes 4 - Indexing

98

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

(d) New root, insert 45

n=3



CS 525



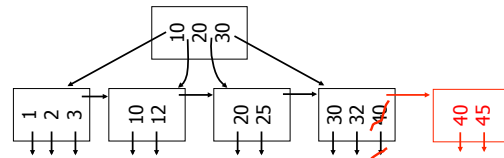
Notes 4 - Indexing

99

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

(d) New root, insert 45

n=3



CS 525



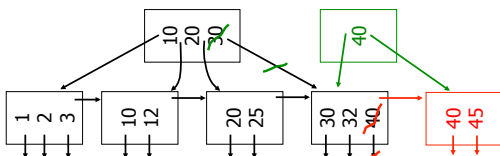
Notes 4 - Indexing

100

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

(d) New root, insert 45

n=3



CS 525



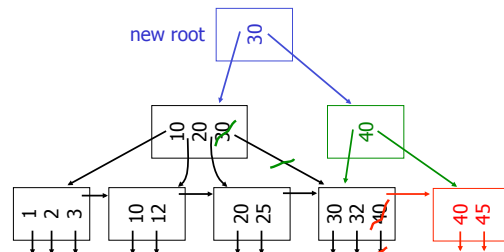
Notes 4 - Indexing

101

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

(d) New root, insert 45

n=3



CS 525



Notes 4 - Indexing

102

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Insertion Algorithm

- Insert Record with key **k**
- Search leaf node for **k**
 - Leaf node has at least one space
 - Insert into leaf
 - Leaf is full
 - Split leaf into two nodes (new leaf)
 - Insert new leaf's smallest key into parent

Insertion Algorithm cont.

- Non-leaf node is full
 - Split parent
 - Insert median key into parent
- Root is full
 - Split root
 - Create new root with two pointers and single key
- -> B-trees grow at the root

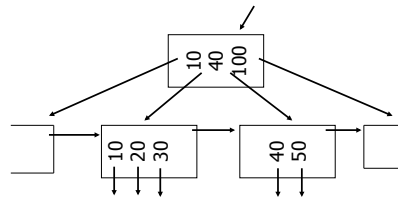
Deletion from B+tree

- Simple case - no example
- Coalesce with neighbor (sibling)
- Re-distribute keys
- Cases (b) or (c) at non-leaf

(b) Coalesce with sibling

– Delete 50

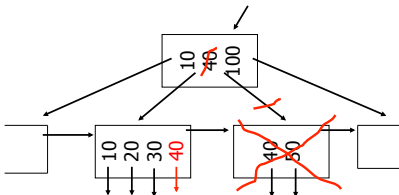
$n=4$



(b) Coalesce with sibling

– Delete 50

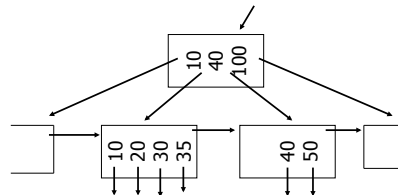
$n=4$



(c) Redistribute keys

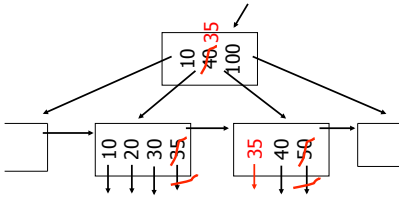
– Delete 50

$n=4$



(c) Redistribute keys
- Delete 50

n=4



CS 525



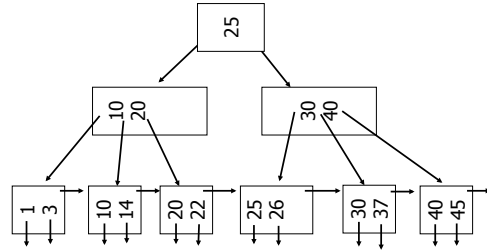
Notes 4 - Indexing

109

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

(d) Non-leaf coalesce
- Delete 37

n=4



CS 525



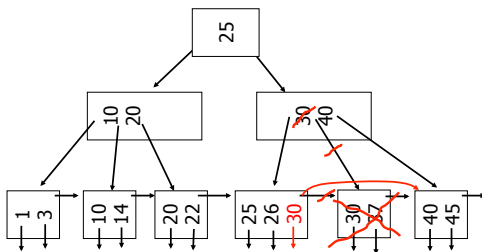
Notes 4 - Indexing

110

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

(d) Non-leaf coalesce
- Delete 37

n=4



CS 525



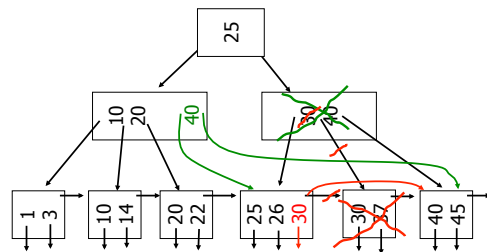
Notes 4 - Indexing

111

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

(d) Non-leaf coalesce
- Delete 37

n=4



CS 525



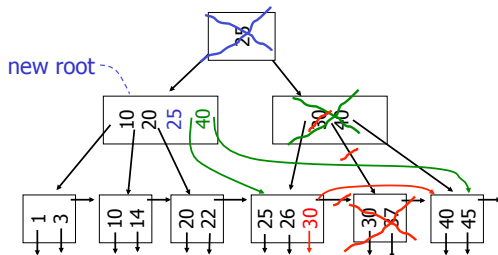
Notes 4 - Indexing

112

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

(d) Non-leaf coalesce
- Delete 37

n=4



CS 525



Notes 4 - Indexing

113

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Deletion Algorithm

- Delete record with key **k**
- Search leaf node for **k**
 - Leaf has more than min entries
 - Remove from leaf
 - Leaf has min entries
 - Try to borrow from sibling
 - One direct sibling has more min entries
 - Move entry from sibling and adapt key in parent

CS 525



Notes 4 - Indexing

114

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Deletion Algorithm cont.

- Both direct siblings have min entries
 - Merge with one sibling
 - Remove node or sibling from parent
 - -> recursive deletion
- Root has two children that get merged
 - Merged node becomes new root

B+tree deletions in practice

- Often, coalescing is not implemented
 - Too hard and not worth it!
 - Assumption: nodes will fill up in time again

Comparison: B-trees vs. static indexed sequential file

Ref #1: Held & Stonebraker
 “B-Trees Re-examined”
 CACM, Feb. 1978

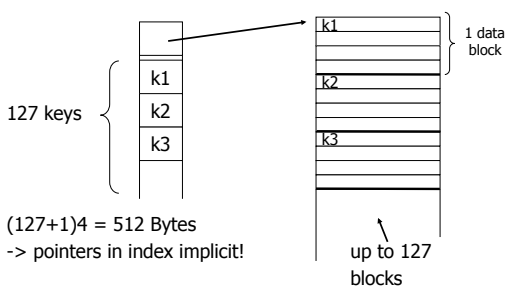
Ref # 1 claims:

- Concurrency control harder in B-Trees
- B-tree consumes more space

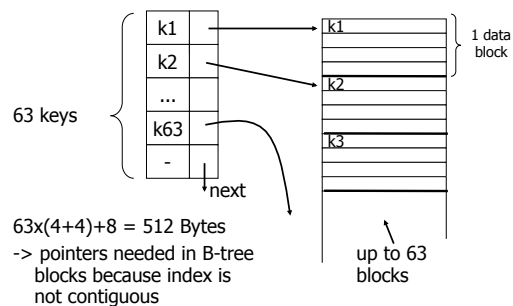
For their comparison:

block = 512 bytes
 key = pointer = 4 bytes
 4 data records per block

Example: 1 block static index



Example: 1 block B-tree



Size comparison

Ref. #1

Static Index		B-tree	
# data blocks	height	# data blocks	height
2 -> 127	2	2 -> 63	2
128 -> 16,129	3	64 -> 3968	3
16,130 -> 2,048,383	4	3969 -> 250,047	4
		250,048 -> 15,752,961	5

CS 525



Notes 4 - Indexing

121



Ref. #1 analysis claims

- For an 8,000 block file,
 - after 32,000 inserts
 - after 16,000 lookups
 ⇒ Static index saves enough accesses to allow for reorganization

CS 525



Notes 4 - Indexing

122



Ref. #1 analysis claims

- For an 8,000 block file,
 - after 32,000 inserts
 - after 16,000 lookups
 ⇒ Static index saves enough accesses to allow for reorganization

Ref. #1 conclusion

Static index better!!

CS 525



Notes 4 - Indexing

123



Ref #2: M. Stonebraker,

“Retrospective on a database system,” TODS, June 1980

Ref. #2 conclusion

B-trees better!!

CS 525



Notes 4 - Indexing

124



Ref. #2 conclusion

B-trees better!!

- DBA does not know when to reorganize
- DBA does not know how full to load pages of new index

Ref. #2 conclusion

B-trees better!!

- Buffering
 - B-tree: has fixed buffer requirements
 - Static index: must read several overflow blocks to be efficient (large & variable buffers size needed for this)

CS 525



Notes 4 - Indexing

125



CS 525



Notes 4 - Indexing

126



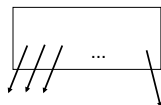
- Speaking of buffering...
Is LRU a good policy for B+tree buffers?

- Speaking of buffering...
Is LRU a good policy for B+tree buffers?

- Of course not!
- Should try to keep root in memory at all times
(and perhaps some nodes from second level)

Interesting problem:

For B+tree, how large should n be?



n is number of keys / node

Sample assumptions:

- (1) Time to read node from disk is $(S+Tn)$ msec.

Sample assumptions:

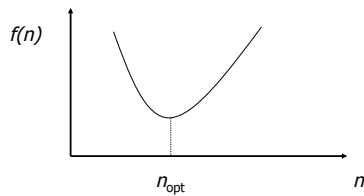
- (1) Time to read node from disk is $(S+Tn)$ msec.
- (2) Once block in memory, use binary search to locate key:
 $(a + b \log_2 n)$ msec.
For some constants a, b ; Assume $a \ll S$

Sample assumptions:

- (1) Time to read node from disk is $(S+Tn)$ msec.
- (2) Once block in memory, use binary search to locate key:
 $(a + b \log_2 n)$ msec.
For some constants a, b ; Assume $a \ll S$
- (3) Assume B+tree is full, i.e.,
nodes to examine is $\log_n N$
where $N = \#$ records

➔ Can get:

$f(n)$ = time to find a record



➔ FIND n_{opt} by $f'(n) = 0$

Answer is n_{opt} = "few hundred"

➔ FIND n_{opt} by $f'(n) = 0$

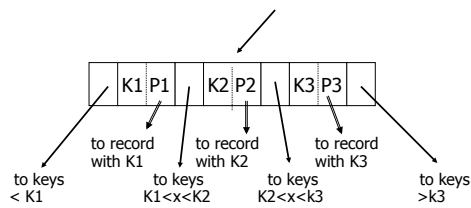
Answer is n_{opt} = "few hundred"

➔ What happens to n_{opt} as

- Disk gets faster?
- CPU get faster?
- Memory hierarchy?

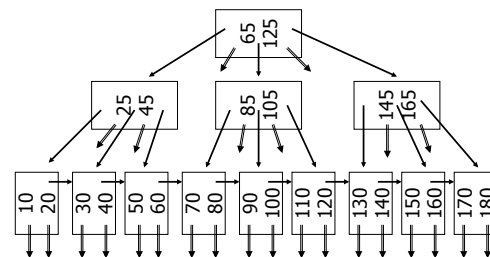
Variation on B+tree: B-tree (no +)

- Idea:
 - Avoid duplicate keys
 - Have record pointers in non-leaf nodes



B-tree example

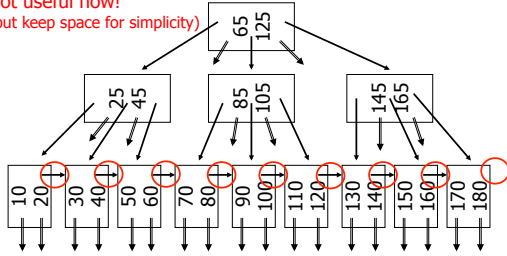
$n=2$



B-tree example

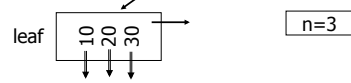
n=2

- sequence pointers
not useful now!
(but keep space for simplicity)



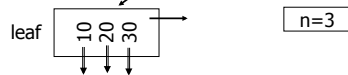
Note on inserts

- Say we insert record with key = 25

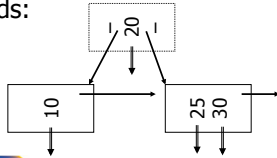


Note on inserts

- Say we insert record with key = 25



- Afterwards:



So, for B-trees:

	MAX			MIN		
	Tree Ptrs	Rec Ptrs	Keys	Tree Ptrs	Rec Ptrs	Keys
Non-leaf non-root	n+1	n	n	$\lceil (n+1)/2 \rceil$	$\lceil (n+1)/2 \rceil - 1$	$\lceil (n+1)/2 \rceil - 1$
Leaf non-root	1	n	n	1	$\lfloor n/2 \rfloor$	$\lfloor n/2 \rfloor$
Root non-leaf	n+1	n	n	2	1	1
Root Leaf	1	n	n	1	1	1

Tradeoffs:

- ☺ B-trees have faster lookup than B+trees
- ☹ in B-tree, non-leaf & leaf different sizes
- ☹ in B-tree, deletion more complicated

Tradeoffs:

- ☺ B-trees have faster lookup than B+trees
- ☹ in B-tree, non-leaf & leaf different sizes
- ☹ in B-tree, deletion more complicated

➡ B+trees preferred!

But note:

- If blocks are fixed size
(due to disk and buffering restrictions)
Then lookup for B+tree is actually better!!

Example:

- Pointers 4 bytes
- Keys 4 bytes
- Blocks 100 bytes (just example)
- Look at full 2 level tree

B-tree:

Root has 8 keys + 8 record pointers
+ 9 son pointers
= $8 \times 4 + 8 \times 4 + 9 \times 4 = 100$ bytes

B-tree:

Root has 8 keys + 8 record pointers
+ 9 son pointers
= $8 \times 4 + 8 \times 4 + 9 \times 4 = 100$ bytes

Each of 9 sons: 12 rec. pointers (+12 keys)
= $12 \times (4+4) + 4 = 100$ bytes

B-tree:

Root has 8 keys + 8 record pointers
+ 9 son pointers
= $8 \times 4 + 8 \times 4 + 9 \times 4 = 100$ bytes

B+tree:

Root has 12 keys + 13 son pointers
= $12 \times 4 + 13 \times 4 = 100$ bytes

Each of 9 sons: 12 rec. pointers (+12 keys)
= $12 \times (4+4) + 4 = 100$ bytes

2-level B-tree, Max # records =
 $12 \times 9 + 8 = 116$

B+tree:

Root has 12 keys + 13 son pointers
= $12 \times 4 + 13 \times 4 = 100$ bytes

Each of 13 sons: 12 rec. ptrs (+12 keys)
= $12 \times (4 + 4) + 4 = 100$ bytes

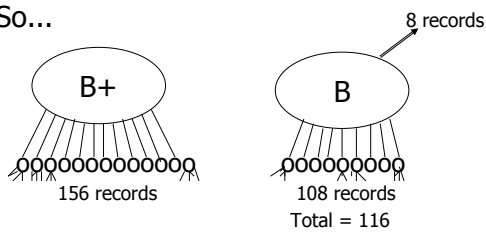
B+tree:

Root has 12 keys + 13 son pointers
= $12 \times 4 + 13 \times 4 = 100$ bytes

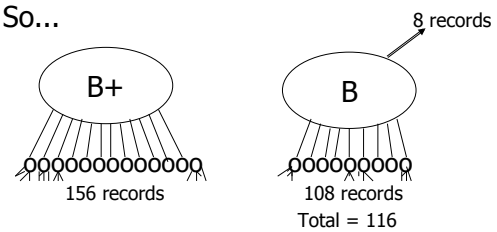
Each of 13 sons: 12 rec. ptrs (+12 keys)
= $12 \times (4 + 4) + 4 = 100$ bytes

2-level B+tree, Max # records
= $13 \times 12 = 156$

So...



So...



Conclusion:

- For fixed block size,
- B+ tree is better because it is bushier

Additional B-tree Variants

- B*-tree
 - Internal nodes have to be 2/3 full

An Interesting Problem...

- What is a good index structure when:
 - records tend to be inserted with keys that are larger than existing values? (e.g., banking records with growing data/time)
 - we want to remove older data

One Solution: Multiple Indexes

- Example: I1, I2

day	days indexed	days indexed
	I1	I2
10	1,2,3,4,5	6,7,8,9,10
11	11,2,3,4,5	6,7,8,9,10
12	11,12,3,4,5	6,7,8,9,10
13	11,12,13,4,5	6,7,8,9,10

- advantage: deletions/insertions from smaller index
- disadvantage: query multiple indexes

Another Solution (Wave Indexes)

day	I1	I2	I3	I4
10	1,2,3	4,5,6	7,8,9	10
11	1,2,3	4,5,6	7,8,9	10,11
12	1,2,3	4,5,6	7,8,9	10,11, 12
13	13	4,5,6	7,8,9	10,11, 12
14	13,14	4,5,6	7,8,9	10,11, 12
15	13,14,15	4,5,6	7,8,9	10,11, 12
16	13,14,15	16	7,8,9	10,11, 12

- advantage: no deletions
- disadvantage: approximate windows

Concurrent Access To B-trees

- Multiple processes/threads accessing the B-tree
 - Can lead to corruption
- Serialize access to complete tree for updates
 - Simple
 - Unnecessary restrictive
 - Not feasible for high concurrency

Lock Nodes

- One solution
 - **Read** and **exclusive** locks
 - Safe and unsafe updates of nodes
 - **Safe:** No ancestor of node will be effected by update
 - **Unsafe:** Ancestor may be affected
 - Can be determined locally
 - E.g., deletion is safe is node has more than n/2

	Read	Write
Read	X	-
Write	-	-

Lock Nodes

- Reading
 - Use standard search algorithm
 - Hold lock on current node
 - Release when navigating to child
- Writing
 - Lock each node on search for key
 - Release all locks on parents of node if the node is safe

Improvements?

- Try locking only the leaf for update
 - Let update use read locks and only lock leaf node with write lock
 - If leaf node is unsafe then use previous protocol
- Many more locking approaches have been proposed

Outline/summary

- Conventional Indexes
 - Sparse vs. dense
 - Primary vs. secondary
- B trees
 - B+trees vs. B-trees
 - B+trees vs. indexed sequential
- Hashing schemes --> Next
- Advanced Index Techniques

CS 525



Notes 4 - Indexing

163

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

CS 525: Advanced Database Organization

05: Hashing and More

Boris Glavic



Slides: adapted from a [course](#) taught by [Hector Garcia-Molina](#), Stanford InfoLab

CS 525



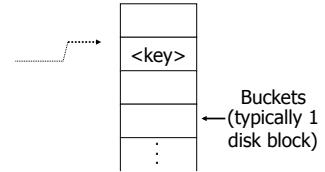
Notes 5 - Hashing

1



Hashing

key \rightarrow h(key)



CS 525

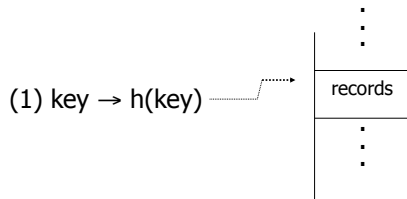


Notes 5 - Hashing

2



Two alternatives



CS 525

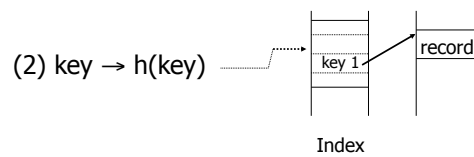


Notes 5 - Hashing

3



Two alternatives



CS 525

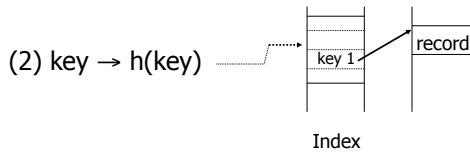


Notes 5 - Hashing

4



Two alternatives



- Alt (2) for "secondary" search key

CS 525



Notes 5 - Hashing

5



Example hash function

- Key = 'x₁ x₂ ... x_n' n byte character string
- Have b buckets
- h: add $x_1 + x_2 + \dots + x_n$
 - compute sum modulo b

CS 525



Notes 5 - Hashing

6



- This may not be best function ...
- Read Knuth Vol. 3 if you really need to select a good function.

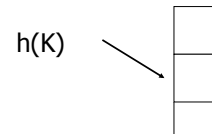
- This may not be best function ...
- Read Knuth Vol. 3 if you really need to select a good function.

Good hash function: ⇨ Expected number of keys/bucket is the same for all buckets

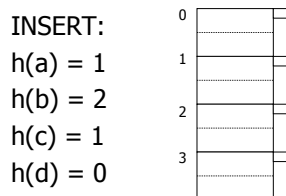
Within a bucket:

- Do we keep keys sorted?
- Yes, if CPU time critical & Inserts/Deletes not too frequent

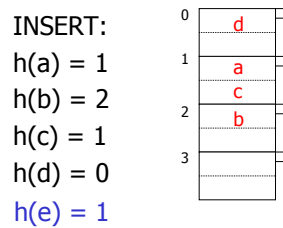
Next: example to illustrate inserts, overflows, deletes



EXAMPLE 2 records/bucket

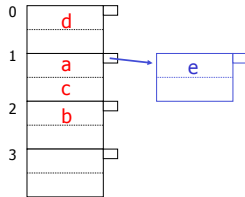


EXAMPLE 2 records/bucket



EXAMPLE 2 records/bucket

INSERT:
 $h(a) = 1$
 $h(b) = 2$
 $h(c) = 1$
 $h(d) = 0$
 $h(e) = 1$



CS 525



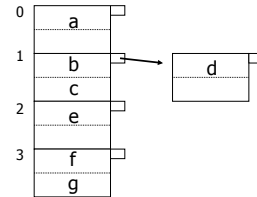
Notes 5 - Hashing

13



EXAMPLE: deletion

Delete:
 e
 f



CS 525



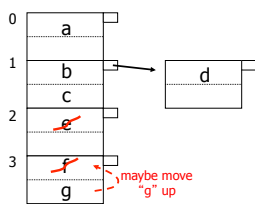
Notes 5 - Hashing

14



EXAMPLE: deletion

Delete:
 e
 f
 c



CS 525



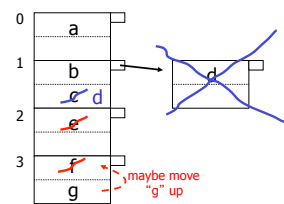
Notes 5 - Hashing

15



EXAMPLE: deletion

Delete:
 e
 f
 c



CS 525



Notes 5 - Hashing

16



Rule of thumb:

- Try to keep space utilization between 50% and 80%

$$\text{Utilization} = \frac{\# \text{ keys used}}{\text{total } \# \text{ keys that fit}}$$

CS 525



Notes 5 - Hashing

17



Rule of thumb:

- Try to keep space utilization between 50% and 80%

$$\text{Utilization} = \frac{\# \text{ keys used}}{\text{total } \# \text{ keys that fit}}$$

- If < 50%, wasting space
- If > 80%, overflows significant
 - ↳ depends on how good hash function is & on # keys/bucket

CS 525



Notes 5 - Hashing

18



How do we cope with growth?

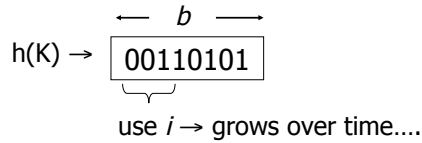
- Overflows and reorganizations
- Dynamic hashing

How do we cope with growth?

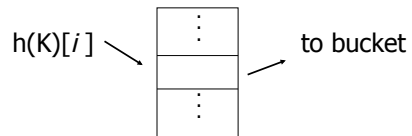
- Overflows and reorganizations
- Dynamic hashing
 - Extensible
 - Linear

Extensible hashing: two ideas

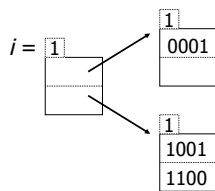
(a) Use i of b bits output by hash function



(b) Use directory

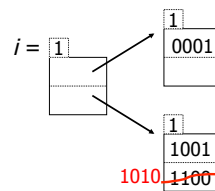


Example: $h(k)$ is 4 bits; 2 keys/bucket



Insert 1010

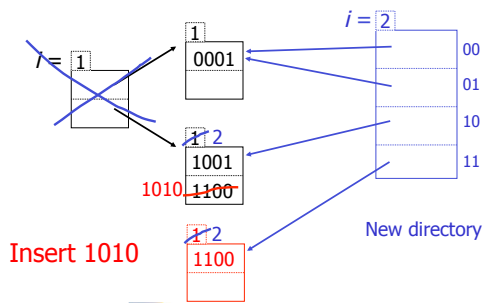
Example: $h(k)$ is 4 bits; 2 keys/bucket



Insert 1010



Example: $h(k)$ is 4 bits; 2 keys/bucket



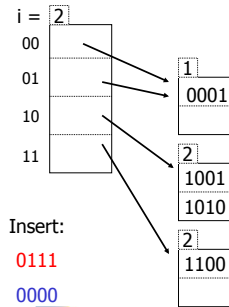
CS 525



Notes 5 - Hashing

25

Example continued



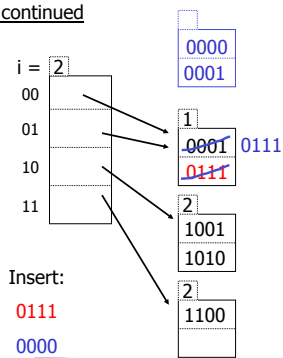
CS 525



Notes 5 - Hashing

26

Example continued



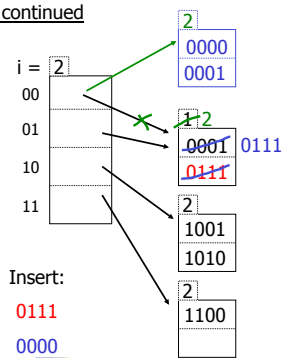
CS 525



Notes 5 - Hashing

27

Example continued



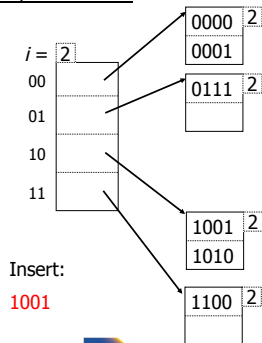
CS 525



Notes 5 - Hashing

28

Example continued



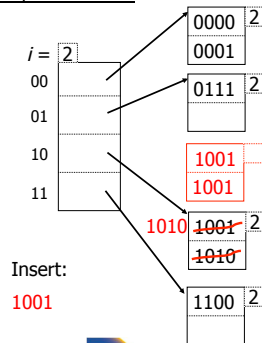
CS 525



Notes 5 - Hashing

29

Example continued



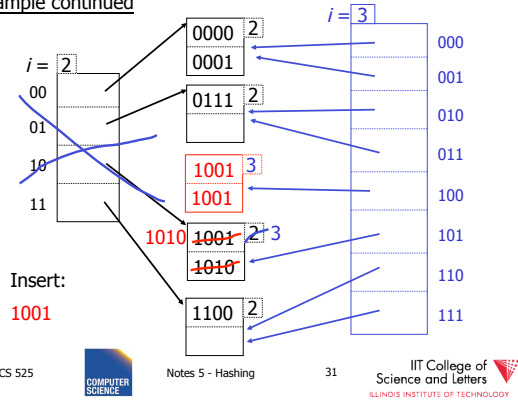
CS 525



Notes 5 - Hashing

30

Example continued



Extensible hashing: deletion

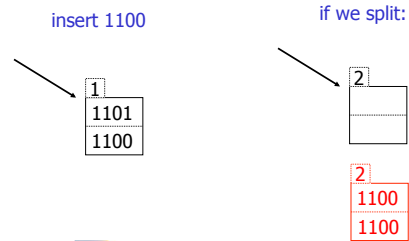
- No merging of blocks
- Merge blocks and cut directory if possible (Reverse insert procedure)

Deletion example:

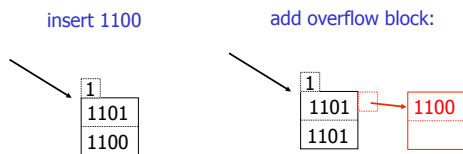
- Run thru insert example in reverse!

Note: Still need overflow chains

- Example: many records with duplicate keys



Solution: overflow chains



Summary Extensible hashing

- ⊕ Can handle growing files
 - with less wasted space
 - with no full reorganizations

Summary Extensible hashing

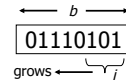
- ⊕ Can handle growing files
 - with less wasted space
 - with no full reorganizations
- ⊖ Indirection
(Not bad if directory in memory)
- ⊖ Directory doubles in size
(Now it fits, now it does not)

Linear hashing

- Another dynamic hashing scheme

Two ideas:

(a) Use i low order bits of hash

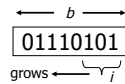


Linear hashing

- Another dynamic hashing scheme

Two ideas:

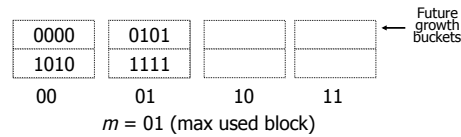
(a) Use i low order bits of hash



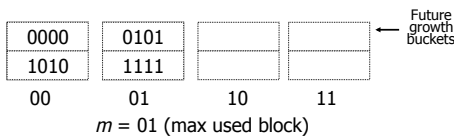
(b) File grows linearly



Example $b=4$ bits, $i=2$, 2 keys/bucket



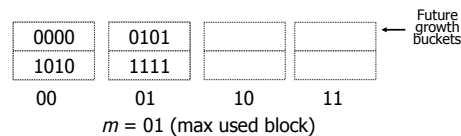
Example $b=4$ bits, $i=2$, 2 keys/bucket



Rule If $h(k)[i] \leq m$, then
 look at bucket $h(k)[i]$
 else, look at bucket $h(k)[i] - 2^{i-1}$

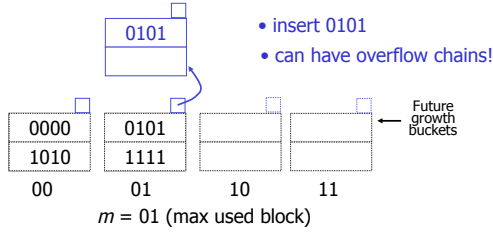
Example $b=4$ bits, $i=2$, 2 keys/bucket

- insert 0101



Rule If $h(k)[i] \leq m$, then
 look at bucket $h(k)[i]$
 else, look at bucket $h(k)[i] - 2^{i-1}$

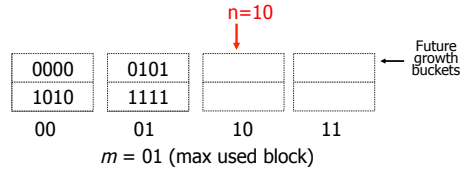
Example $b=4$ bits, $i=2$, 2 keys/bucket



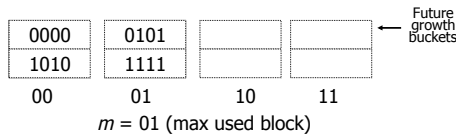
Rule If $h(k)[i] \leq m$, then look at bucket $h(k)[i]$ else, look at bucket $h(k)[i] - 2^{i-1}$

Note

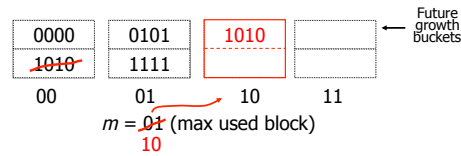
- In textbook, n is used instead of m
- $n = m + 1$



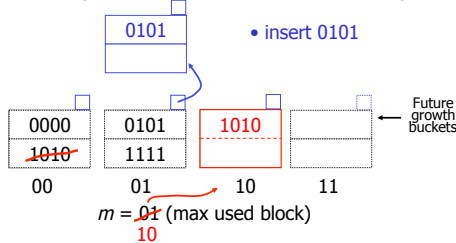
Example $b=4$ bits, $i=2$, 2 keys/bucket



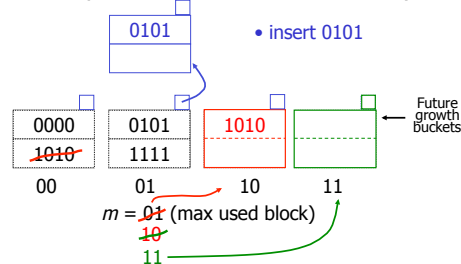
Example $b=4$ bits, $i=2$, 2 keys/bucket



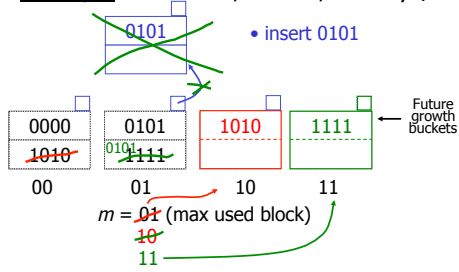
Example $b=4$ bits, $i=2$, 2 keys/bucket



Example $b=4$ bits, $i=2$, 2 keys/bucket



Example $b=4$ bits, $i=2$, 2 keys/bucket



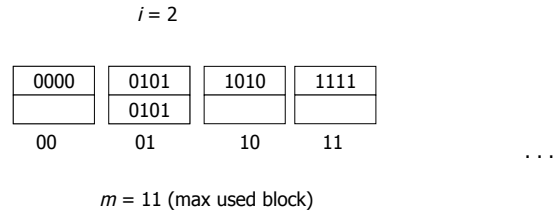
CS 525



Notes 5 - Hashing

49

Example Continued: How to grow beyond this?



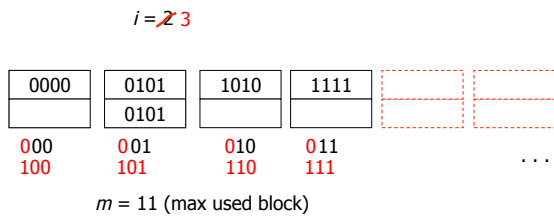
CS 525



Notes 5 - Hashing

50

Example Continued: How to grow beyond this?



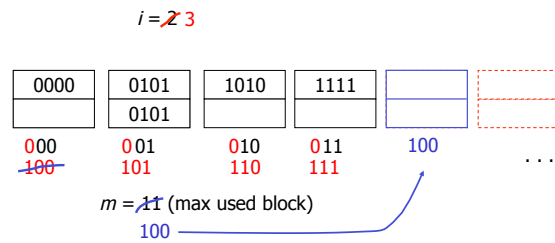
CS 525



Notes 5 - Hashing

51

Example Continued: How to grow beyond this?



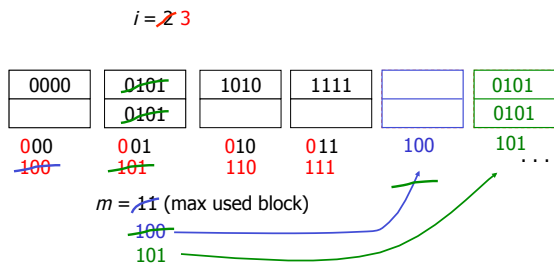
CS 525



Notes 5 - Hashing

52

Example Continued: How to grow beyond this?



CS 525



Notes 5 - Hashing

53

• When do we expand file?

• Keep track of: $\frac{\# \text{ used slots}}{\text{total \# of slots}} = U$

CS 525



Notes 5 - Hashing

54

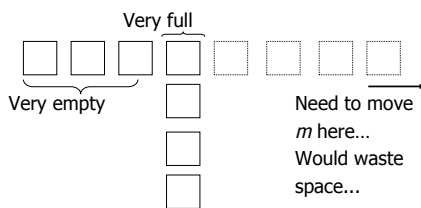
☛ When do we expand file?

- Keep track of: $\frac{\text{\# used slots}}{\text{total \# of slots}} = U$
- If $U > \text{threshold}$ then increase m
(and maybe i)

Summary Linear Hashing

- ⊕ Can handle growing files
 - with less wasted space
 - with no full reorganizations
- ⊕ No indirection like extensible hashing
- ⊖ Can still have overflow chains

Example: BAD CASE



Summary

Hashing

- How it works
- Dynamic hashing
 - Extensible
 - Linear

Next:

- Indexing vs Hashing
- Index definition in SQL
- Multiple key access

Indexing vs Hashing

- Hashing good for probes given key
e.g.,
SELECT ...
FROM R
WHERE R.A = 5
-> **Point Queries**

Indexing vs Hashing

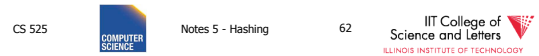
- INDEXING (Including B Trees) good for Range Searches:
e.g.,
SELECT
FROM R
WHERE R.A > 5

-> Range Queries



Index definition in SQL

- Create index name on rel (attr)
- Create unique index name on rel (attr)
 - ↳ defines candidate key
- Drop INDEX name



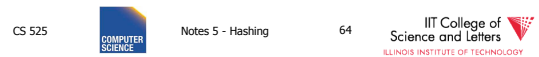
Note CANNOT SPECIFY TYPE OF INDEX
(e.g. B-tree, Hashing, ...)
OR PARAMETERS
(e.g. Load Factor, Size of Hash,...)

... at least in standard SQL...

Vendor specific extensions allow that



Note ATTRIBUTE LIST ⇒ MULTIKEY INDEX
(next)
e.g., CREATE INDEX foo ON R(A,B,C)

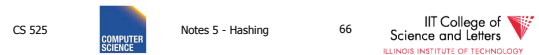
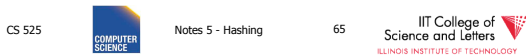
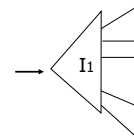


Multi-key Index

Motivation: Find records where
DEPT = "Toy" AND SAL > 50k

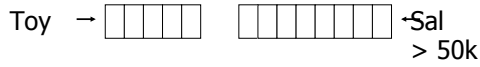
Strategy I:

- Use one index, say Dept.
- Get all Dept = "Toy" records and check their salary



Strategy II:

- Use 2 Indexes; Manipulate Pointers



CS 525



Notes 5 - Hashing

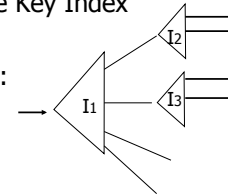
67

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Strategy III:

- Multiple Key Index

One idea:



CS 525

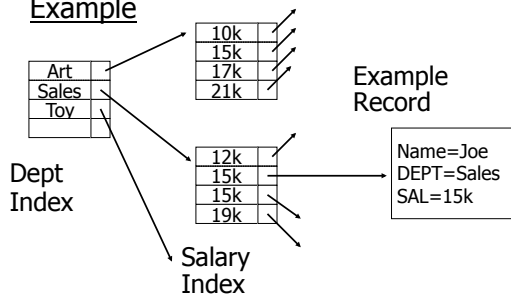


Notes 5 - Hashing

68

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Example



CS 525



Notes 5 - Hashing

69

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

For which queries is this index good?

- Find RECs Dept = "Sales" \wedge SAL=20k
- Find RECs Dept = "Sales" \wedge SAL \geq 20k
- Find RECs Dept = "Sales"
- Find RECs SAL = 20k

CS 525



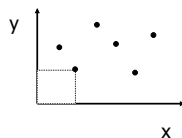
Notes 5 - Hashing

70

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Interesting application:

- Geographic Data



DATA:
 $\langle X_1, Y_1, \text{Attributes} \rangle$
 $\langle X_2, Y_2, \text{Attributes} \rangle$
 \vdots

CS 525



Notes 5 - Hashing

71

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Queries:

- What city is at $\langle X_i, Y_i \rangle$?
- What is within 5 miles from $\langle X_i, Y_i \rangle$?
- Which is closest point to $\langle X_i, Y_i \rangle$?

CS 525

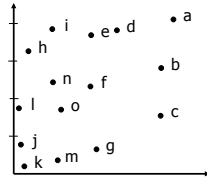


Notes 5 - Hashing

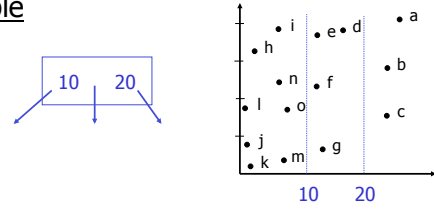
72

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Example



Example



CS 525



Notes 5 - Hashing

73



CS 525

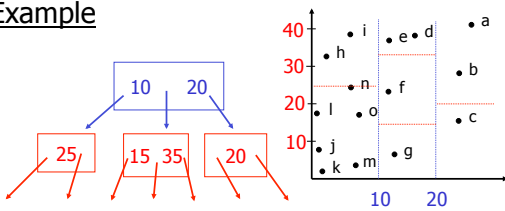


Notes 5 - Hashing

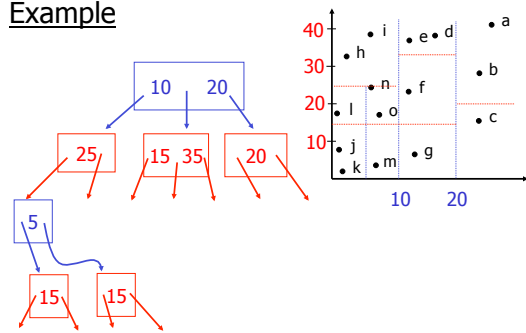
74



Example



Example



CS 525



Notes 5 - Hashing

75



CS 525

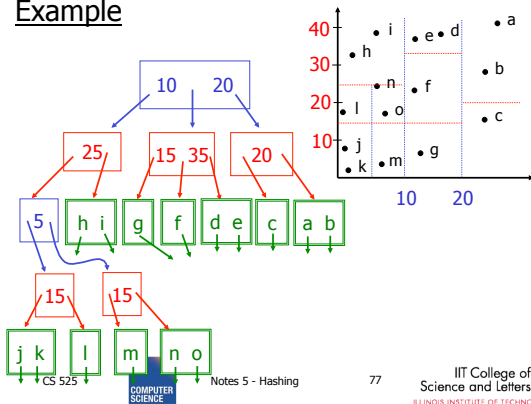


Notes 5 - Hashing

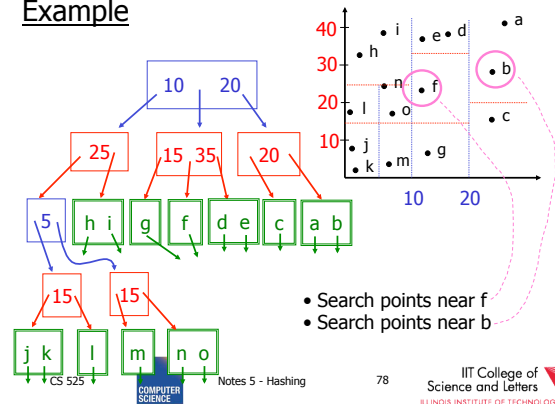
76



Example



Example



CS 525



Notes 5 - Hashing

77



CS 525



Notes 5 - Hashing

78



Queries

- Find points with $Y_i > 20$
- Find points with $X_i < 5$
- Find points “close” to $i = \langle 12, 38 \rangle$
- Find points “close” to $b = \langle 7, 24 \rangle$

Next

- Even more index structures ☺

CS 525



Notes 5 - Hashing

79



CS 525



Notes 5 - Hashing


80



CS 525: Advanced Database Organization

06: Even more index structures

Boris Glavic



Slides: adapted from a [course](#) taught by [Hector Garcia-Molina](#), Stanford InfoLab

CS 525 COMPUTER SCIENCE Notes 6 - More Indices 1 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

Recap

- We have discussed
 - Conventional Indices
 - B-trees
 - Hashing
 - Trade-offs
 - Multi-key indices
 - Multi-dimensional indices
 - ... but no example

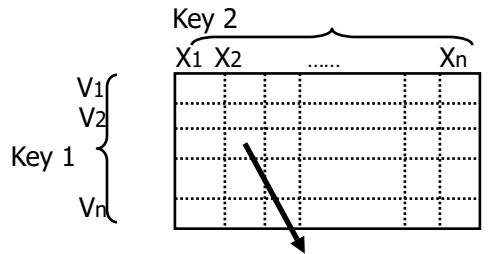
CS 525 COMPUTER SCIENCE Notes 6 - More Indices 2 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

Today

- **Multi-dimensional index structures**
 - kd-Trees (very similar to example before)
 - **Grid File (Grid Index)**
 - Quad Trees
 - **R Trees**
 - **Partitioned Hash**
 - ...
- **Bitmap-indices**
- **Tries**

CS 525 COMPUTER SCIENCE Notes 6 - More Indices 3 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

Grid Index



To records with key1=V3, key2=X2

CS 525 COMPUTER SCIENCE Notes 5 - Hashing 4 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

CLAIM

- Can quickly find records with
 - key 1 = $V_i \wedge$ Key 2 = X_j
 - key 1 = V_i
 - key 2 = X_j

CS 525 COMPUTER SCIENCE Notes 5 - Hashing 5 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

CLAIM

- Can quickly find records with
 - key 1 = $V_i \wedge$ Key 2 = X_j
 - key 1 = V_i
 - key 2 = X_j
- And also ranges....
 - E.g., key 1 $\geq V_i \wedge$ key 2 $< X_j$

CS 525 COMPUTER SCIENCE Notes 5 - Hashing 6 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

• How do we find entry i, j in linear structure?

max number of i values $N=4$

$pos(i, j) =$

0,0	position S+0
0,1	position S+1
0,2	position S+2
0,3	position S+3
1,0	position S+4
1,1	
1,2	
1,3	
2,0	
2,1	position S+9
2,2	
2,3	
3,0	

CS 525 COMPUTER SCIENCE Notes 5 - Hashing 7 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

• How do we find entry i, j in linear structure?

max number of i values $N=4$

$pos(i, j) = S + iN + j$

Issue: Cells must be same size, and N must be constant!

Issue: Some cells may overflow, some may be sparse...

0,0	position S+0
0,1	position S+1
0,2	position S+2
0,3	position S+3
1,0	position S+4
1,1	
1,2	
1,3	
2,0	
2,1	position S+9
2,2	
2,3	
3,0	

CS 525 COMPUTER SCIENCE Notes 5 - Hashing 8 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

Solution: Use Indirection

Buckets

V1	X1	X2	X3	...
V2				...
V3				...
V4				...

*Grid only contains pointers to buckets

Buckets

CS 525 COMPUTER SCIENCE Notes 5 - Hashing 9 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

With indirection:

- Grid can be regular without wasting space
- We do have price of indirection

CS 525 COMPUTER SCIENCE Notes 5 - Hashing 10 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

Can also index grid on value ranges

Salary Grid

0-20K	1
20K-50K	2
50K-∞	3

Linear Scale

1	2	3
Toy	Sales	Personnel

CS 525 COMPUTER SCIENCE Notes 5 - Hashing 11 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

Grid files

- ⊕ Good for multiple-key search
- ⊖ Space, management overhead (nothing is free)
- ⊖ Need partitioning ranges that evenly split keys

CS 525 COMPUTER SCIENCE Notes 5 - Hashing 12 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

Partitioned hash function

Idea:

010110 1110010

Key1 → (h1) (h2) ← Key2

CS 525 COMPUTER SCIENCE Notes 5 - Hashing 13 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

EX:

h1(toy) =0	000	
h1(sales) =1	001	
h1(art) =1	010	
.	011	
.	100	
h2(10k) =01	101	
h2(20k) =11	110	
h2(30k) =01	111	
h2(40k) =00		

Insert → <Fred,toy,10k>, <Joe,sales,10k>
<Sally,art,30k>

CS 525 COMPUTER SCIENCE Notes 5 - Hashing 14 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

EX:

h1(toy) =0	000	
h1(sales) =1	001	<Fred>
h1(art) =1	010	
.	011	
.	100	
h2(10k) =01	101	<Joe><Sally>
h2(20k) =11	110	
h2(30k) =01	111	
h2(40k) =00		

Insert → <Fred,toy,10k>, <Joe,sales,10k>
<Sally,art,30k>

CS 525 COMPUTER SCIENCE Notes 5 - Hashing 15 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

EX:

h1(toy) =0	000	<Fred>
h1(sales) =1	001	<Joe><Jan>
h1(art) =1	010	<Mary>
.	011	
.	100	<Sally>
h2(10k) =01	101	
h2(20k) =11	110	<Tom><Bill>
h2(30k) =01	111	<Andy>
h2(40k) =00		

Find Emp. with Dept. = Sales \wedge Sal=40k

CS 525 COMPUTER SCIENCE Notes 5 - Hashing 16 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

EX:

h1(toy) =0	000	<Fred>
h1(sales) =1	001	<Joe><Jan>
h1(art) =1	010	<Mary>
.	011	
.	100	<Sally>
h2(10k) =01	101	
h2(20k) =11	110	<Tom><Bill>
h2(30k) =01	111	<Andy>
h2(40k) =00		

Find Emp. with Dept. = Sales \wedge Sal=40k

CS 525 COMPUTER SCIENCE Notes 5 - Hashing 17 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

EX:

h1(toy) =0	000	<Fred>
h1(sales) =1	001	<Joe><Jan>
h1(art) =1	010	<Mary>
.	011	
.	100	<Sally>
h2(10k) =01	101	
h2(20k) =11	110	<Tom><Bill>
h2(30k) =01	111	<Andy>
h2(40k) =00		



Find Emp. with Sal=30k

CS 525 COMPUTER SCIENCE Notes 5 - Hashing 18 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

EX:

h1(toy) =0	000	<Fred>
h1(sales) =1	001	<Joe><Jan>
h1(art) =1	010	<Mary>
.	011	
.	100	<Sally>
h2(10k) =01	101	<Tom><Bill>
h2(20k) =11	110	<Tom><Bill>
h2(30k) =01	111	<Andy>
h2(40k) =00		



Find Emp. with Sal=30k

CS 525  Notes 5 - Hashing 19  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

EX:

h1(toy) =0	000	<Fred>
h1(sales) =1	001	<Joe><Jan>
h1(art) =1	010	<Mary>
.	011	
.	100	<Sally>
h2(10k) =01	101	<Tom><Bill>
h2(20k) =11	110	<Tom><Bill>
h2(30k) =01	111	<Andy>
h2(40k) =00		



Find Emp. with Dept. = Sales

CS 525  Notes 5 - Hashing 20  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

EX:



h1(toy) =0	000	<Fred>
h1(sales) =1	001	<Joe><Jan>
h1(art) =1	010	<Mary>
.	011	
.	100	<Sally>
h2(10k) =01	101	<Tom><Bill>
h2(20k) =11	110	<Tom><Bill>
h2(30k) =01	111	<Andy>
h2(40k) =00		

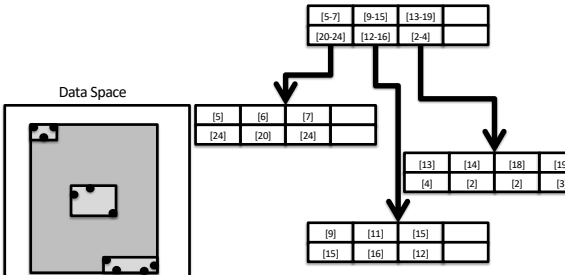
Find Emp. with Dept. = Sales



CS 525  Notes 5 - Hashing 21  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

R-tree

- Nodes can store up to **M** entries
 - Minimum fill requirement (depends on variant)
- Each node rectangle in **n**-dimensional space
 - Minimum Bounding Rectangle (MBR) of its children
- MBRs of siblings are allowed to overlap
 - Different from B-trees
- balanced

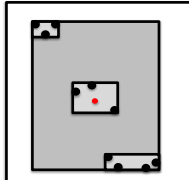
CS 525  Notes 6 - More Indices 22  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY





CS 525  Notes 6 - More Indices 23  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

R-tree - Search

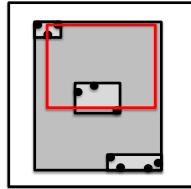
- Point Search
 - Search for $p = \langle x_i, y_i \rangle$
 - Keep list of potential nodes
 - Needed because of overlap
 - Traverse to child if MBR of child contains p



CS 525  Notes 6 - More Indices 24  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

R-tree - Search

- Point Search
 - Search for points in region = $\langle [x_{\min} - x_{\max}], [y_{\min} - y_{\max}] \rangle$
 - Keep list of potential nodes
 - Traverse to child if MBR of child overlaps with query region



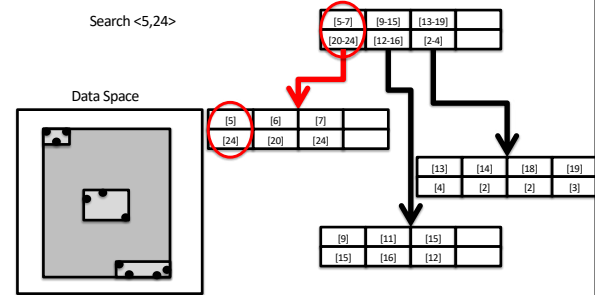
CS 525



Notes 6 - More Indices

25

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Search $\langle 5, 24 \rangle$ 

CS 525



Notes 6 - More Indices

26

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

R-tree - Insert

- Similar to B-tree, but more complex
 - Overlap \rightarrow multiple choices where to add entry
 - Split harder because more choice how to split node (compare B-tree = 1 choice)
- 1) Find potential subtrees for current node
 - Choose one for insert (heuristic, e.g., the one the would grow the least)
 - Continue until leaf is found

CS 525



Notes 6 - More Indices

27

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

R-tree - Insert

- 2) Insert into leaf
- 3) Leaf is full? \rightarrow split
 - Find best split (minimum overlap between new nodes) is hard ($O(2^M)$)
 - Use linear or quadratic heuristics (original paper)
- 4) Adapt parents if necessary

CS 525



Notes 6 - More Indices

28

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

R-tree - Delete

- 1) Find leaf node that contains entry
- 2) Delete entry
- 3) Leaf node underflow?
 - Remove leaf node and cache entries
 - Adapt parents
 - Reinsert deleted entries

CS 525



Notes 6 - More Indices

29

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Bitmap Index

- Domain of values $D = \{d_1, \dots, d_n\}$
 - Gender {male, female}
 - Age {1, ..., 120?}
- Use one vector of bits for each value
 - One bit for each record
 - 0: record has different value in this attribute
 - 1: record has this value

CS 525



Notes 6 - More Indices

30

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Bitmap Index Example

Age		
1	2	3
1	0	0
0	1	0
1	0	0
0	0	1

Todlers		
Name	Age	Gender
Peter	1	male
Gertrud	2	female
Joe	1	male
Marry	3	female

Gender	
male	female
1	0
0	1
1	0
0	1

CS 525 Notes 6 - More Indices 31

Bitmap Index Example

Age		
1	2	3
1	0	0
0	1	0
1	0	0
0	0	1

Todlers		
Name	Age	Gender
Peter	1	male
Gertrud	2	female
Joe	1	male
Marry	3	female

Gender	
male	female
1	0
0	1
1	0
0	1

Find all todlers with age **2** and sex **female**:
Bitwise-and between vectors

0
1
0
0

CS 525 Notes 6 - More Indices 32

Bitmap Index Example

Age		
1	2	3
1	0	0
0	1	0
1	0	0
0	0	1

Todlers		
Name	Age	Gender
Peter	1	male
Gertrud	2	female
Joe	1	male
Marry	3	female

Gender	
male	female
1	0
0	1
1	0
0	1

Find all todlers with age **2** or sex **female**:
Bitwise-or between vectors

0
1
0
1

CS 525 Notes 6 - More Indices 33

Compression

- Observation:
 - Each record has one value in indexed attribute
 - For N records and domain of size |D|
 - Only 1/|D| bits are 1
 - -> waste of space
- Solution
 - Compress data
 - Need to make sure that **and** and **or** is still fast

CS 525 Notes 6 - More Indices 34

Run length encoding (RLE)

- Instead of actual 0-1 sequence encode length of 0 or 1 runs
- One bit to indicate whether 0/1 run + several bits to encode run length
- But how many bits to use to encode a run length?
 - Gamma codes or similar to have variable number of bits

CS 525 Notes 6 - More Indices 35

RLE Example

- 0001 0000 1110 1111 **(2 bytes)**
- **3, 1,4, 3, 1,4** **(6 bytes)**
- -> if we use one byte to encode a run we have 7 bits for length = max run length is 128(127)

CS 525 Notes 6 - More Indices 36

Elias Gamma Codes

- $X = 2^N + (x \bmod 2^N)$
 - Write N as N zeros followed by one 1
 - Write $(x \bmod 2^N)$ as N bit number
- $18 = 2^4 + 2 = 000010010$
- 0001 0000 1110 1111 **(2 bytes)**
- 3, 1,4, 3, 1,4 **(6 bytes)**
- 0111 0010 0011 1001 00 **(3 bytes)**

CS 525



Notes 6 - More Indices

37

Hybrid Encoding

- Run length encoding
 - Can waste space
 - And/or run length not aligned to byte/word boundaries
- Encode some bytes of sequence as is and only store long runs as run length
 - EWAH
 - BBC (that's what Oracle uses)

CS 525

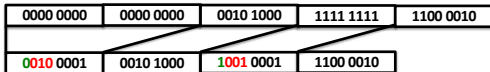


Notes 6 - More Indices

38

Extended Word aligned Hybrid (EWAH)

- Segment sequence in machine words (64bit)
- Use two types of words to encode
 - Literal words, taken directly from input sequence
 - Run words
 - $\frac{1}{2}$ word is used to encode a run
 - $\frac{1}{2}$ word is used to encode how many literals follow



CS 525



Notes 6 - More Indices

39

Bitmap Indices

- Fast for read intensive workloads
 - Used a lot in data warehousing
- Often build on the fly during query processing
 - As we will see later in class

CS 525



Notes 6 - More Indices

40

Trie

- From Retrieval
- Tree index structure
- Keys are sequences of values from a domain D
 - $D = \{0,1\}$
 - $D = \{a,b,c,\dots,z\}$
- Key size may or may not be fixed
 - Store 4-byte integers using $D = \{0,1\}$ (32 elements)
 - Strings using $D = \{a,\dots,z\}$ (arbitrary length)

CS 525



Notes 6 - More Indices

41

Trie

- Each node has pointers to $|D|$ child nodes
 - One for each value of D
- Searching for a key $k = [d_1, \dots, d_n]$
 - Start at the root
 - Follow child for value d_i

CS 525




Notes 6 - More Indices

42

Trie Example


Words: bar, ball, in

Search for **bold**


Notes 6 - More Indices 43
IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY


Tries Implementation

- 1) Each node has an array of child pointers
- 2) Each node has a list or hash table of child pointers
- 3) array compression schemes derived from compressed DFA representations


Notes 6 - More Indices 44
IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY


Index structures in the Main Memory DBMS era

- Larger and large portions of the data fit into main memory
 - Disk I/O no longer the (only) bottleneck
 - Highly optimized and specialized operator code
 - Difference of the constant factor for full scan versus index increase
 - Increasing amounts of parallelism
 - Traditional methods for parallel access to indexes no longer effective enough
- => Do not use indexes anymore?


Notes 6 - More Indices 45
IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY


Index structures in the Main Memory DBMS era

- Solutions
 - More Light-weight and coarse-grained data structures
 - Use data-structures that have less parallelization bottle-necks


Notes 6 - More Indices 46
IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Index structures in the Main Memory DBMS era


- **Solutions**
 - More Light-weight and coarse-grained data structures, e.g.:
 - Data skipping (small materialized aggregates)
 - Database cracking
 - Use data-structures that have less parallelization bottle-necks, e.g.,
 - Skip lists
 - B^w-trees


Notes 6 - More Indices 47
IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Data skipping

- Consider a relation stored in an unsorted page file
 - Regular DBMS
 - HDFS parquet file
 - ...
- Main idea of data skipping
 - For each page store min/max values of each attribute
- To evaluate a selection predicate on attribute A say $c1 \leq A \leq c2$
 - if for page P: $A_{max} < c1$ or $A_{min} > c2$ then none of the tuples on that page will qualify and we can skip reading this page

R			
	A	B	C
a	1	10	
b	5	20	
c	2	10	
d	2	35	
e	3	45	
f	4	40	


Notes 6 - More Indices 48
IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Database cracking

- Main rationale
 - Originally designed for columnar databases
 - The amount of indexing effort we spend for a part of the key space should be based on how frequently this part of the key space is accessed
- Basic idea
 - Start with an unsorted file
 - Whenever a query applies a selection condition on a column A, say $A < 50$, then split the current partition containing 50 into two fragments one with data < 50 and one with the remaining data (partial sort)
 - Keep a small in-memory tree index for these fragments

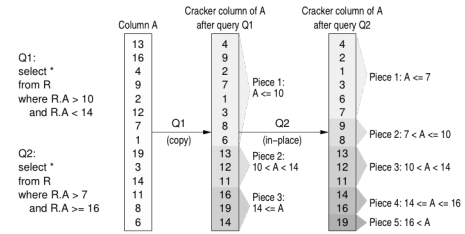
CS 525



Notes 6 - More Indices 49



Database cracking



From Database Cracking – CIDR 2007

CS 525



Notes 6 - More Indices 50



Skip lists

- Probabilistic datastructure
 - Behavior depends on randomization
 - Gives only probabilistic guarantees
 - => with high probability will guarantee good performance
 - Approximates a search tree using the much simpler (and easier to parallelize linked list datastructure)

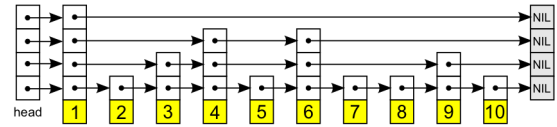
CS 525



Notes 6 - More Indices 51



Skip lists



- Search:
 - Start from the top list
 - 1) Move through list until element is found or we are at a larger element/end of the list
 - 2) move to previous element (smaller than search key) and follow a down pointer to the next deeper level
 - 3) Goto 1)

CS 525

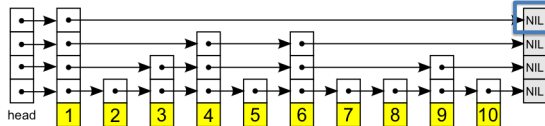


Notes 6 - More Indices 52



Search 5

Skip lists



- Search:
 - Start from the top list
 - 1) Move through list until element is found or we are at a larger element/end of the list
 - 2) move to previous element (smaller than search key) and follow a down pointer to the next deeper level
 - 3) Goto 1)

CS 525

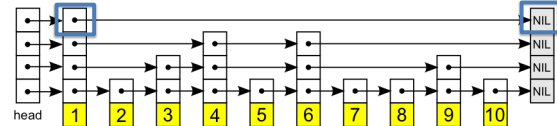


Notes 6 - More Indices 53



Search 5

Skip lists



- Search:
 - Start from the top list
 - 1) Move through list until element is found or we are at a larger element/end of the list
 - 2) move to previous element (smaller than search key) and follow a down pointer to the next deeper level
 - 3) Goto 1)

CS 525



Notes 6 - More Indices 54



Search 5

Skip lists

- **Search:**
 - Start from the top list
 - 1) Move through list until element is found or we are at a larger element/end of the list
 - 2) move to previous element (smaller than search key) and follow a down pointer to the next deeper level
 - 3) Goto 1)

CS 525 COMPUTER SCIENCE Notes 6 - More Indices 55 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

Search 5

Skip lists

- **Search:**
 - Start from the top list
 - 1) Move through list until element is found or we are at a larger element/end of the list
 - 2) move to previous element (smaller than search key) and follow a down pointer to the next deeper level
 - 3) Goto 1)

CS 525 COMPUTER SCIENCE Notes 6 - More Indices 56 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

Search 5

Skip lists

- **Search:**
 - Start from the top list
 - 1) Move through list until element is found or we are at a larger element/end of the list
 - 2) move to previous element (smaller than search key) and follow a down pointer to the next deeper level
 - 3) Goto 1)

CS 525 COMPUTER SCIENCE Notes 6 - More Indices 57 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

Search 5

Skip lists

- **Search:**
 - Start from the top list
 - 1) Move through list until element is found or we are at a larger element/end of the list
 - 2) move to previous element (smaller than search key) and follow a down pointer to the next deeper level
 - 3) Goto 1)

CS 525 COMPUTER SCIENCE Notes 6 - More Indices 58 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

Search 5

Skip lists

- **Search:**
 - Start from the top list
 - 1) Move through list until element is found or we are at a larger element/end of the list
 - 2) move to previous element (smaller than search key) and follow a down pointer to the next deeper level
 - 3) Goto 1)

CS 525 COMPUTER SCIENCE Notes 6 - More Indices 59 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

Skip lists

- **Insert:**
 - Use search to find insertion position at the lowest level (keep pointers at the higher levels)
 - Insert element in the lowest list
 - Then for every level throw a dice and insert key with probability p (typically $\frac{1}{2}$)

Observation: in expectation each level has p as many nodes as the next lower level

CS 525 COMPUTER SCIENCE Notes 6 - More Indices 60 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

Skip lists

- **Characteristics**
 - $O(\log(n))$ expected performance (insert, delete, search)
 - Easy to parallelize (linked lists)
 - Simpler to implement (also less CPU ops) than B-trees
- **Example implementations**
 - MemSQL (main memory database system)
 - Lucene
 - levelldb

CS 525



Notes 6 - More Indices 61

Improving insert/update performance

- B-tree
 - $O(\log(n))$ I/O
- Hash-index
 - $O(1)$ I/O, but potential reorg cost
- Consider Key-value store (e.g., Cassandra) application
 - Need fast write-throughput
 - Need fast point-lookup

CS 525



Notes 6 - More Indices 62

One Solution: LSM-trees

- **Log-structured merge (LSM) trees**
 - Have small index that is memory resident (**memtable**)
 - When memtable exceeds a size threshold write it as one sorted run to disk (will explain algorithm when talking about query execution)
 - Sequential I/O!
 - Runs are immutable after being written (exception compaction)
 - Runs may contain outdated values for keys that exist in newer runs of the memtable
 - Over time we have multiple sorted runs
 - **Inserts/Updates**
 - Always applied to memtable
 - **Lookup**
 - If we find a key in the memtable then return it
 - Otherwise lookup keys in the sorted runs in reverse chronological order

CS 525



Notes 6 - More Indices 63

LSM-trees

- **Performance**
 - **Inserts/Updates**
 - $O(1)!$
 - **Lookup**
 - $O(\#runs)$
 - => want to make sure the number of runs does not grow indefinitely
- **Compaction**
 - Merge sorted runs on disks to reduce $\#runs$ => improve lookup performance

CS 525



Notes 6 - More Indices 64

Basic Compaction

- Have levels
 - Once there are more than x runs on a level these are merged into one larger run
 - Run sizes increase exponentially per level
- E.g., threshold is 4 runs
 - first level: runs are of same size as memtable
 - 2nd level: $4 * \text{size of memtable}$
 - 3rd level: $4 * 4 * \text{size of memtable}$
 - ...

CS 525



Notes 6 - More Indices 65

LSM-trees

- **Other lookup improvements**
 - Block index in memory (similar to sparse index)
 - Bloomfilters
 - A bloom filter is a small over-approximation of set
 - Can be used to test if a key K is contained in a set S
 - » Returns yes, then the key **may** be in the set
 - » Returns no, then the key is guaranteed to not be in the set
 - => fast way to avoid looking at runs that are guaranteed to not contain a key

CS 525



Notes 6 - More Indices 66

Bw-trees

• Motivation

- Improve concurrency properties of B-trees
- Improve cache effectiveness of B-trees
- Designed for flash-storage
 - Fast random/sequential reads
 - Fast sequential writes
 - Comparably slower random writes (albeit smaller factor)

CS 525



Notes 6 - More Indices 67

Bw-trees

• Overview

- Updateable B-tree without latches
 - Threads almost never block
 - => Improved instruction cache performance
- Backed up by log-structured storage
- Updates never modify pages but append deltas to a page
 - Deltas are "installed" using CAS (atomic compare and swap)

CS 525

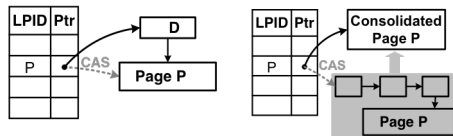


Notes 6 - More Indices 68

Bw-trees

• Mapping table

- Pages are logical identified by a LPID which is stable
- Locations and size of pages can change over time
- Updates create a delta record that points to the previous address of the page
- The delta record's address is swapped for the current address in the mapping table using CAS



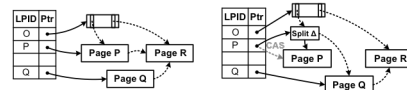
CS 525



Notes 6 - More Indices 69

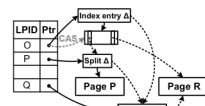
Bw-trees

• Making page splits atomic



(a) Creating sibling page Q

(b) Installing split delta



(c) Installing index entry delta

CS 525



Notes 6 - More Indices 70

Summary

Discussion:

- Conventional Indices
- B-trees
- Hashing (extensible, linear)
- SQL Index Definition
- Index vs. Hash
- Multiple Key Access
- Multi Dimensional Indices
 - Variations: Grid, R-tree,
- Partitioned Hash
- Bitmap indices and compression
- Tries
- Database cracking
- Data skipping (small materialized aggregates/zone maps)
- Skip-lists
- Log-structured merge trees (LSM)

CS 525



Notes 5 - Hashing 71

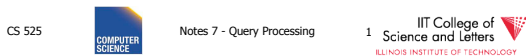


CS 525: Advanced Database Organisation

07: Query Processing Overview

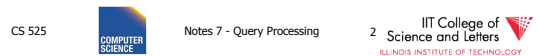
Boris Glavic

Slides: adapted from a [course](#) taught by [Hector Garcia-Molina](#), Stanford InfoLab



Query Processing

Q → Query Plan



Query Processing

Q → Query Plan

Focus: Relational Systems

- Others?

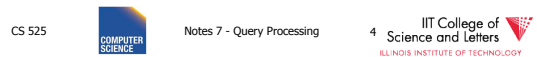
Example

Select B,D

From R,S

Where $R.A = "c" \wedge S.E = 2$

$R.C = S.C$

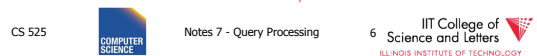
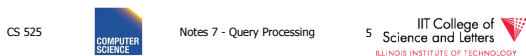


R	A	B	C	S	C	D	E
	a	1	10		10	x	2
	b	1	20		20	y	2
	c	2	10		30	z	2
	d	2	35		40	x	1
	e	3	45		50	y	3

R	A	B	C	S	C	D	E
	a	1	10		10	x	2
	b	1	20		20	y	2
	c	2	10		30	z	2
	d	2	35		40	x	1
	e	3	45		50	y	3

Answer

B	D
2	x



- How do we execute query?

One idea

- Do Cartesian product
- Select tuples
- Do projection

RXS	R.A	R.B	R.C	S.C	S.D	S.E
	a	1	10	10	x	2
	a	1	10	20	y	2
	⋮					
	C	2	10	10	x	2
	⋮					

CS 525



Notes 7 - Query Processing



CS 525



Notes 7 - Query Processing



RXS	R.A	R.B	R.C	S.C	S.D	S.E
	a	1	10	10	x	2
	a	1	10	20	y	2
	⋮					
Bingo! → Got one...	C	2	10	10	x	2
	⋮					

CS 525

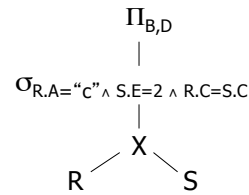


Notes 7 - Query Processing



Relational Algebra - can be used to describe plans...

Ex: Plan I



CS 525

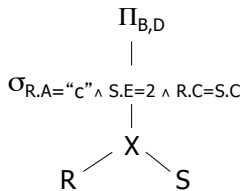


Notes 7 - Query Processing



Relational Algebra - can be used to describe plans...

Ex: Plan I



OR: $\Pi_{B,D} [\sigma_{R.A='c' \wedge S.E=2 \wedge R.C=S.C} (RXS)]$

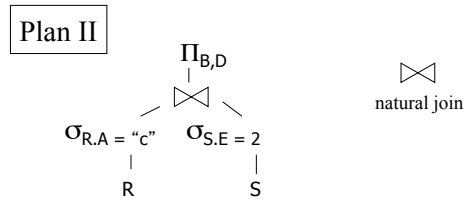
CS 525



Notes 7 - Query Processing



Another idea:

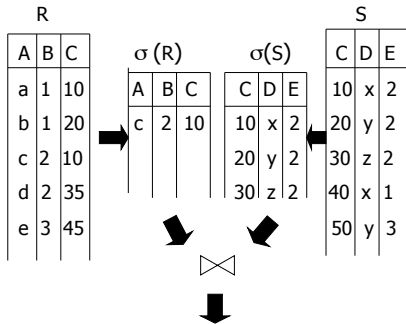


CS 525



Notes 7 - Query Processing





Plan III

Use R.A and S.C Indexes

- (1) Use R.A index to select R tuples with R.A = "c"
- (2) For each R.C value found, use S.C index to find matching tuples

CS 525



Notes 7 - Query Processing

13



CS 525



Notes 7 - Query Processing

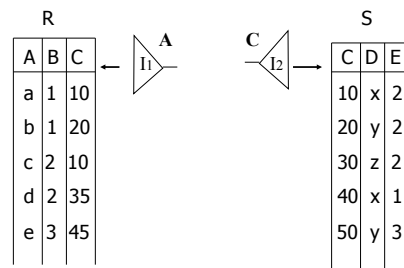
14



Plan III

Use R.A and S.C Indexes

- (1) Use R.A index to select R tuples with R.A = "c"
- (2) For each R.C value found, use S.C index to find matching tuples
- (3) Eliminate S tuples S.E \neq 2
- (4) Join matching R,S tuples, project B,D attributes and place in result



CS 525



Notes 7 - Query Processing

15

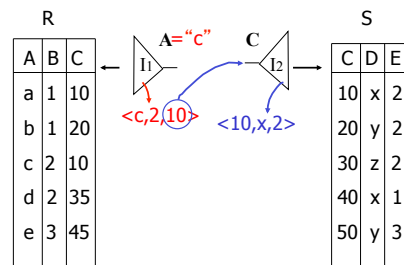
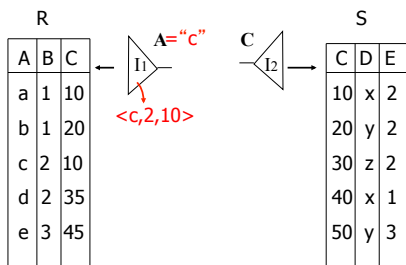


CS 525



Notes 7 - Query Processing

16



CS 525



Notes 7 - Query Processing

17



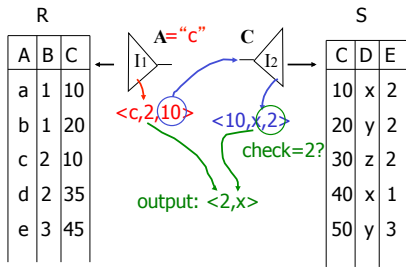
CS 525



Notes 7 - Query Processing

18





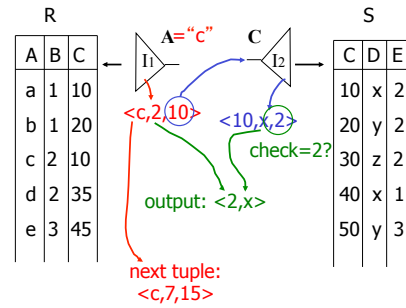
CS 525



Notes 7 - Query Processing

19

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY



CS 525

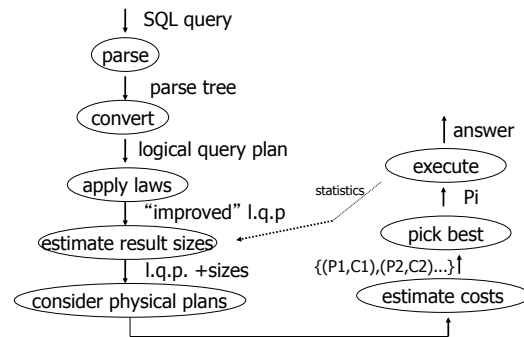


Notes 7 - Query Processing

20

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Overview of Query Optimization



CS 525



Notes 7 - Query Processing

21

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

CS 525



Notes 7 - Query Processing

22

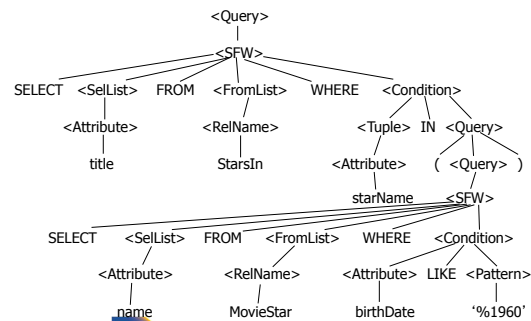
IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Example: SQL query

```
SELECT title
FROM StarsIn
WHERE starName IN (
  SELECT name
  FROM MovieStar
  WHERE birthdate LIKE '%1960'
);
```

(Find the movies with stars born in 1960)

Example: Parse Tree



CS 525



Notes 7 - Query Processing

23

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

CS 525



Notes 7 - Query Processing

24

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Example: Generating Relational Algebra

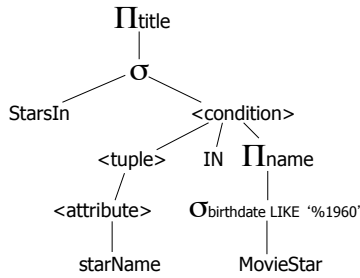


Fig. 7.15: An expression using a two-argument σ , midway between a parse tree and relational algebra

CS 525



Notes 7 - Query Processing

25



Example: Logical Query Plan

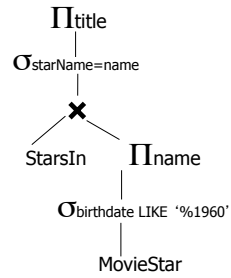


Fig. 7.18: Applying the rule for IN conditions

CS 525

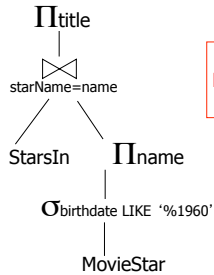


Notes 7 - Query Processing

26



Example: Improved Logical Query Plan



Question:
Push project to
StarsIn?

Fig. 7.20: An improvement on fig. 7.18.

CS 525

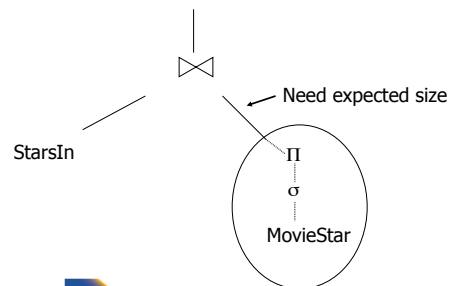


Notes 7 - Query Processing

27



Example: Estimate Result Sizes



CS 525

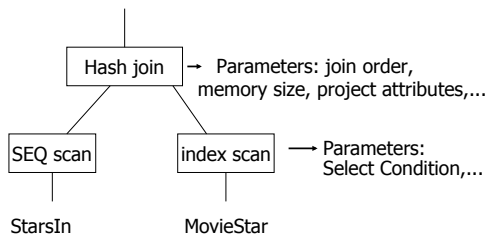


Notes 7 - Query Processing

28



Example: One Physical Plan



CS 525

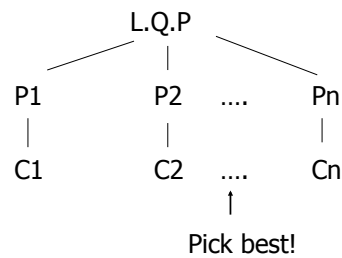


Notes 7 - Query Processing

29



Example: Estimate costs



CS 525



Notes 7 - Query Processing

30





CS 525: Advanced Database Organisation

08: Query Processing Parsing and Analysis

Boris Glavic

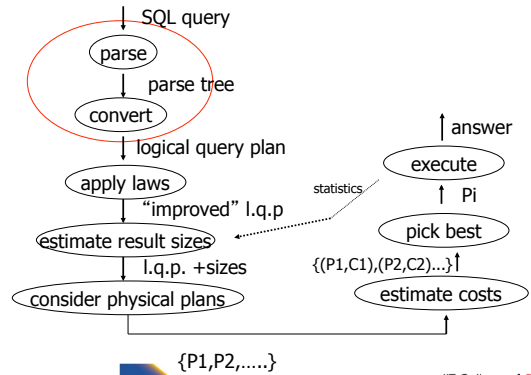
Slides: adapted from a [course](#) taught by [Hector Garcia-Molina](#), Stanford InfoLab

CS 525



Notes 8 - Parsing and Analysis

1



CS 525



Notes 8 - Parsing and Analysis

2



Parsing, Analysis, Conversion

1. Parsing
 - Transform SQL text into syntax tree
2. Analysis
 - Check for semantic correctness
 - Use database catalog
 - E.g., unfold views, lookup functions and attributes, check scopes
3. Conversion
 - Transform into internal representation
 - Relational algebra or QBM

CS 525



Notes 8 - Parsing and Analysis

3



Analysis and Conversion

- Usually intertwined
- The internal representation is used to store analysis information
- Create an initial representation and complete during analysis

CS 525



Notes 8 - Parsing and Analysis

4



Parsing, Analysis, Conversion

1. Parsing
2. Analysis
3. Conversion

Parsing

- SQL -> Parse Tree
- Covered in compiler courses and books
- Here only short overview

CS 525



Notes 8 - Parsing and Analysis

5



CS 525



Notes 8 - Parsing and Analysis

6



SQL Standard

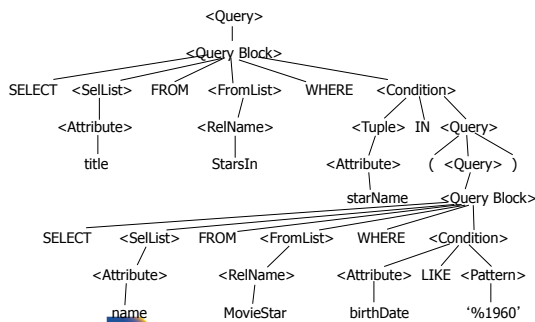
- Standardized language
 - 86, 89, 92, 99, 03, 06, 08, 11
- DBMS vendors developed their own dialects

Example: SQL query

```
SELECT title
FROM StarsIn
WHERE starName IN (
    SELECT name
    FROM MovieStar
    WHERE birthdate LIKE '%1960'
);
```

(Find the movies with stars born in 1960)

Example: Parse Tree



SQL Query Structure

- Organized in Query blocks
- ```
SELECT <select_list>
FROM <from_list>
WHERE <where_condition>
GROUP BY <group_by_expressions>
HAVING <having_condition>
ORDER BY <order_by_expressions>
```

## Query Blocks

- Only **SELECT** clause is mandatory
  - Some DBMS require **FROM**

**SELECT** (1 + 2) AS result

|        |
|--------|
| result |
| 3      |



## SELECT clause

- List of expressions and optional name assignment + optional **DISTINCT**
  - Attribute references: R.a, b
  - Constants: 1, 'hello', '2008-01-20'
  - Operators: (R.a + 3) \* 2
  - Functions (maybe UDF): substr(R.a, 1,3)
    - Single result or **set functions**
  - Renaming: (R.a + 2) AS x

## SELECT clause - example

```
SELECT substring(p.name,1,1) AS initial
 p.name
FROM person p
```

| person |        | result  |      |
|--------|--------|---------|------|
| name   | gender | initial | name |
| Joe    | male   | J       | Joe  |
| Jim    | male   | J       | Jim  |

CS 525  Notes 8 - Parsing and Analysis 13  IIT College of Science and Letters  
ILLINOIS INSTITUTE OF TECHNOLOGY



## SELECT clause – set functions

- Function `extrChar(string)`

```
SELECT extrChar(p.name) AS n
FROM person p
```

| person |        |
|--------|--------|
| name   | gender |
| Joe    | male   |
| Jim    | male   |



| result |
|--------|
| n      |
| J      |
| o      |
| e      |
| J      |
| i      |
| m      |

CS 525  Notes 8 - Parsing and Analysis 14  IIT College of Science and Letters  
ILLINOIS INSTITUTE OF TECHNOLOGY

## SELECT clause – DISTINCT



```
SELECT DISTINCT gender
FROM person p
```

| person |        | result |
|--------|--------|--------|
| name   | gender | gender |
| Joe    | male   | male   |
| Jim    | male   |        |

CS 525  Notes 8 - Parsing and Analysis 15  IIT College of Science and Letters  
ILLINOIS INSTITUTE OF TECHNOLOGY



## FROM clause

- List of table expressions
  - Access to relations
  - Subqueries (need alias)
  - Join expressions
  - Table functions
  - Renaming of relations and columns

CS 525  Notes 8 - Parsing and Analysis 16  IIT College of Science and Letters  
ILLINOIS INSTITUTE OF TECHNOLOGY



## FROM clause examples

```
FROM R
 -access table R
FROM R, S
 -access tables R and S
FROM R JOIN S ON (R.a = S.b)
 -join tables R and S on condition (R.a = S.b)
FROM R x
FROM R AS x
 -Access table R and assign alias 'x'
```

CS 525  Notes 8 - Parsing and Analysis 17  IIT College of Science and Letters  
ILLINOIS INSTITUTE OF TECHNOLOGY

## FROM clause examples

```
FROM R x(c,d)
FROM R AS x(c,d)
 -using aliases x for R and c,d for its attributes
FROM (R JOIN S t ON (R.a = t.b)), T
 -join R and S, and access T
FROM (R JOIN S ON (R.a = S.b)) JOIN T
 -join tables R and S and result with T
FROM create_sequence(1,100) AS seq(a)
 -call table function
```

CS 525  Notes 8 - Parsing and Analysis 18  IIT College of Science and Letters  
ILLINOIS INSTITUTE OF TECHNOLOGY

## FROM clause examples

```
FROM
 (SELECT count(*) FROM employee)
 AS empcnt(cnt)
```

-count number of employee in subquery

## FROM clause examples

```
SELECT *
FROM create_sequence(1,3) AS seq(a)
```

**result**

| a |
|---|
| 1 |
| 2 |
| 3 |

## FROM clause examples

```
SELECT dep, headcnt
FROM (SELECT count(*) AS headcnt, dep
 FROM employee
 GROUP BY dep)
WHERE headcnt > 100
```

**employee**

| name | dep       |
|------|-----------|
| Joe  | IT        |
| Jim  | Marketing |
| ...  | ...       |

**result**

| dep     | headcnt |
|---------|---------|
| IT      | 103     |
| Support | 2506    |
| ...     | ...     |

## FROM clause - correlation

- Correlation
  - Reference attributes from other FROM clause item
  - Attributes of  $i^{\text{th}}$  entry only available in  $j > i$
  - Semantics:
    - For each row in result of  $i^{\text{th}}$  entry:
    - Substitute correlated attributes with value from current row and evaluate query

## Correlation - Example

```
SELECT name, chr
FROM employee AS e,
 extrChar(e.name) AS c(chr)
```

**employee**

| name | dep       |
|------|-----------|
| Joe  | IT        |
| Jim  | Marketing |
| ...  | ...       |

**result**

| name | chr |
|------|-----|
| Joe  | J   |
| Joe  | o   |
| Joe  | e   |
| Jim  | J   |
| Jim  | i   |
| ...  | ... |

## Correlation - Example

```
SELECT name
FROM (SELECT max(salary) maxsal
 FROM employee) AS m,
 (SELECT name
 FROM employee x
 WHERE x.salary = m.maxsal) AS e
```

**employee**



| name | salary |
|------|--------|
| Joe  | 20,000 |
| Jim  | 30,000 |
| ...  | ...    |

**result**

| name |
|------|
| Jim  |



## WHERE clause

- A condition
  - Attribute references
  - Constants
  - Operators (boolean)
  - Functions
  - Nested subquery expressions
- Result has to be boolean

CS 525  Notes 8 - Parsing and Analysis 25  IIT College of Science and Letters  
ILLINOIS INSTITUTE OF TECHNOLOGY



## WHERE clause examples

- WHERE  $R.a = 3$   
-comparison between attribute and constant
- WHERE  $(R.a > 5) \text{ AND } (R.a < 10)$   
-range query using boolean AND
- WHERE  $R.a = S.b$   
-comparison between two attributes
- WHERE  $(R.a * 2) > (S.b - 3)$   
-using operators

CS 525  Notes 8 - Parsing and Analysis 26  IIT College of Science and Letters  
ILLINOIS INSTITUTE OF TECHNOLOGY



## Nested Subqueries

- Nesting a query within an expression
- Correlation allowed
  - Access FROM clause attributes
- Different types of nesting
  - Scalar subquery
  - Existential quantification
  - Universal quantification

CS 525  Notes 8 - Parsing and Analysis 27  IIT College of Science and Letters  
ILLINOIS INSTITUTE OF TECHNOLOGY

## Nested Subqueries Semantics



- For each tuple produced by the FROM clause execute the subquery
  - If correlated attributes replace them with tuple values

CS 525  Notes 8 - Parsing and Analysis 28  IIT College of Science and Letters  
ILLINOIS INSTITUTE OF TECHNOLOGY

## Scalar subquery

- Subquery that returns one result tuple
  - How to check?
  - -> Runtime error



```
SELECT *
FROM R
WHERE R.a = (SELECT count(*) FROM S)
```

CS 525  Notes 8 - Parsing and Analysis 29  IIT College of Science and Letters  
ILLINOIS INSTITUTE OF TECHNOLOGY

## Existential Quantification

- $\langle \text{expr} \rangle \text{ IN } \langle \text{subquery} \rangle$ 
  - Evaluates to true if  $\langle \text{expr} \rangle$  equal to at least one of the results of the subquery



```
SELECT *
FROM users
WHERE name IN (SELECT name FROM
blacklist)
```

CS 525  Notes 8 - Parsing and Analysis 30  IIT College of Science and Letters  
ILLINOIS INSTITUTE OF TECHNOLOGY

## Existential Quantification

- EXISTS <subquery>
  - Evaluates to true if <subquery> returns at least one tuple



```
SELECT *
FROM users u
WHERE EXISTS (SELECT * FROM
 blacklist b
 WHERE b.name = u.name)
```

CS 525  Notes 8 - Parsing and Analysis 31  IIT College of Science and Letters  
ILLINOIS INSTITUTE OF TECHNOLOGY

## Existential Quantification

- <expr> <op> ANY <subquery>
  - Evaluates to true if <expr> <op> <tuple> evaluates to true for **at least one** result tuple
  - Op is any comparison operator: =, <, >, ...



```
SELECT *
FROM users
WHERE name = ANY (SELECT name FROM
 blacklist)
```

CS 525  Notes 8 - Parsing and Analysis 32  IIT College of Science and Letters  
ILLINOIS INSTITUTE OF TECHNOLOGY

## Universal Quantification

- <expr> <op> ALL <subquery>
  - Evaluates to true if <expr> <op> <tuple> evaluates to true for **all** result tuples
  - Op is any comparison operator: =, <, >, ...

```
SELECT *
FROM nation
WHERE nname = ALL (SELECT nname FROM
 blacklist)
```



CS 525  Notes 8 - Parsing and Analysis 33  IIT College of Science and Letters  
ILLINOIS INSTITUTE OF TECHNOLOGY

## Nested Subqueries Example

```
SELECT dep,name
FROM employee e
WHERE salary >= ALL (SELECT salary
 FROM employee d
 WHERE e.dep = d.dep)
```



| employee |     |        |
|----------|-----|--------|
| name     | dep | salary |
| Joe      | IT  | 2000   |
| Jim      | IT  | 300    |
| Bob      | HR  | 100    |
| Alice    | HR  | 10000  |
| Patrice  | HR  | 10000  |

| result |         |
|--------|---------|
| dep    | Name    |
| IT     | Joe     |
| HR     | Alice   |
| HR     | Patrice |

CS 525  Notes 8 - Parsing and Analysis 34  IIT College of Science and Letters  
ILLINOIS INSTITUTE OF TECHNOLOGY



## GROUP BY clause

- A list of expressions
  - Same as WHERE
  - No restriction to boolean
  - DBMS has to know how to compare = for data type
- Results are grouped by values of the expressions
- -> usually used for aggregation

CS 525  Notes 8 - Parsing and Analysis 35  IIT College of Science and Letters  
ILLINOIS INSTITUTE OF TECHNOLOGY

## GROUP BY restrictions

- If group-by is used then
  - SELECT clause can only use group by expressions or aggregation functions

CS 525  Notes 8 - Parsing and Analysis 36  IIT College of Science and Letters  
ILLINOIS INSTITUTE OF TECHNOLOGY

## GROUP BY clause examples

- GROUP BY R.a
  - group on single attribute
- GROUP BY (1+2)
  - allowed but useless (single group)
- GROUP BY salary / 1000
  - groups of salary values in buckets of 1000
- GROUP BY R.a, R.b
  - group on two attributes

```
SELECT count(*) AS numP,
 (SELECT count(*)
 FROM friends o
 WHERE o.with = f.name) AS numF
FROM (SELECT DISTINCT name FROM friends) f
GROUP BY (SELECT count(*)
 FROM friends o
 WHERE o.with = f.name)
```

**result**

| numP | numF |
|------|------|
| 1    | 1    |
| 2    | 2    |

**friends**

| name  | with  |
|-------|-------|
| Joe   | Jim   |
| Joe   | Peter |
| Jim   | Joe   |
| Jim   | Peter |
| Peter | Joe   |

## HAVING clause

- A boolean expression
- Applied after grouping and aggregation
  - Only references aggregation expressions and group by expressions

## HAVING clause examples

```
...
HAVING sum(R.a) > 100
 -only return tuples with sum bigger than 100

...
GROUP BY dep
HAVING dep = 'IT' AND sum(salary) > 1000000
 -only return group 'IT' and sum threshold
```

## ORDER BY clause

- A list of expressions
- Semantics: Order the result on these expressions

## ORDER BY clause examples

```
ORDER BY R.a ASC
ORDER BY R.a
 -order ascending on R.a
ORDER BY R.a DESC
 -order descending on R.a
ORDER BY salary + bonus
 -order by sum of salary and bonus
```

## New and Non-standard SQL features (excerpt)

- LIMIT / OFFSET
  - Only return a fix maximum number of rows
  - FETCH FIRST n ROWS ONLY (DB2)
  - row\_number() (Oracle)
- Window functions
  - More flexible grouping
  - Return both aggregated results and input values

CS 525



Notes 8 - Parsing and Analysis

43



## Parsing, Analysis, Conversion

1. Parsing
2. Analysis
3. Conversion

CS 525



Notes 8 - Parsing and Analysis

44



## Analysis Goals

- Semantic checks
  - Table column exists
  - Operator, function exists
  - Determine type casts
  - Scope checks
- Rewriting
  - Unfolding views

CS 525



Notes 8 - Parsing and Analysis

45



## Semantic checks

- ```
SELECT *  
FROM R  
WHERE R.a + 3 > 5
```
- Table R exists?
 - Expand *: which attributes in R?
 - R.a is a column?
 - Type of constants 3, 5?
 - Operator + for types of R.a and 3 exists?
 - Operator > for types of result of + and 5 exists?

CS 525



Notes 8 - Parsing and Analysis

46



Database Catalog

- Stores information about database objects
- Aliases:
 - Information Schema
 - System tables
 - Data Dictionary

CS 525



Notes 8 - Parsing and Analysis

47



Typical Catalog Information

- Tables
 - Name, attributes + data types, constraints
- Schema, DB
 - Hierarchical structuring of data
- Data types
 - Comparison operators
 - physical representation
 - Functions to (de)serialize to string

CS 525





Notes 8 - Parsing and Analysis

48





Typical Catalog Information

- Functions (including aggregate/set)
 - Build-in
 - User defined (UDF)
- Triggers
- Stored Procedures
- ...

CS 525  Notes 8 - Parsing and Analysis 49  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY



Type Casts

- Similar to automatic type conversion in programming languages
- Expression: $R.a + 3.0$
 - Say $R.a$ is of type integer
 - Search for a function $+(int, float)$
 - Does not exist?
 - Try to find a way to cast $R.a$, 3.0 or both to new data type
 - So that a function $+$ exists for new types

CS 525  Notes 8 - Parsing and Analysis 50  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY



Scope checks

- Check that references are in correct scope
- E.g., if GROUP BY is present then SELECT clause expression can only reference group by expressions or aggregated values

CS 525  Notes 8 - Parsing and Analysis 51  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

View Unfolding



- SQL allows for stored queries using CREATE VIEW
- Afterwards a view can be used in queries
- If view is not materialized, then need to replace view with its definition

CS 525  Notes 8 - Parsing and Analysis 52  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

View Unfolding Example

```
CREATE VIEW totalSalary AS
SELECT name, salary + bonus AS total
FROM employee
```



```
SELECT *
FROM totalSalary
WHERE total > 10000
```

CS 525  Notes 8 - Parsing and Analysis 53  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

View Unfolding Example

```
CREATE VIEW totalSalary AS
SELECT name, salary + bonus AS total
FROM employee
```

```
SELECT *
FROM (SELECT name,
            salary + bonus AS total
      FROM employee) AS totalSalary
WHERE total > 10000
```

CS 525  Notes 8 - Parsing and Analysis 54  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Analysis Summary

- Perform semantic checks
 - Catalog lookups (tables, functions, types)
 - Scope checks
- View unfolding
- Generate internal representation during analysis

Parsing, Analysis, Conversion

1. Parsing
2. Analysis
3. Conversion

Conversion

- Create an internal representation
 - Should be useful for analysis
 - Should be useful optimization
- Internal representation
 - Relational algebra
 - Query tree/graph models
 - E.g., QGM (Query Graph Model) in Starburst

Relational Algebra

- Formal language
- Good for studying logical optimization and query equivalence (containment)
- Not informative enough for analysis
 - No datatype representation in algebra expressions
 - No meta-data

Other Internal Representations

- Practical implementations
 - Mostly following structure of SQL query blocks
 - Store data type and meta-data (where necessary)

Canonical Translation to Relational Algebra

- TEXTBOOK version of conversion
- Given an SQL query
- Return an equivalent relational algebra expression

Relational Algebra Recap

- Formal query language
- Consists of operators
 - Input(s): relation
 - Output: relation
 - -> Composable
- Set and Bag semantics version

- Relation Schema
 - A set of attribute name-datatype pairs
- Relation (instance)
 - A (multi-)set of tuples with the same schema
- Tuple
 - List of attribute value pairs (or function from attribute name to value)

Set- vs. Bag semantics

- Set semantics:
 - Relations are Sets
 - Used in most theoretical work
- Bag semantics
 - Relations are Multi-Sets
 - Each element (tuple) can appear more than once
 - SQL uses bag semantics

Bag semantics notation

- We use \mathbf{t}^m to denote tuple t appears with multiplicity m

Set- vs. Bag semantics

Set		Bag	
Name	Purchase	Name	Purchase
Peter	Guitar	Peter	Guitar
Joe	Drum	Peter	Guitar
Alice	Bass	Joe	Drum
		Alice	Bass
		Alice	Bass

Operators

- Selection
- Renaming
- Projection
- Joins
 - Theta, natural, cross-product, outer, anti
- Aggregation
- Duplicate removal
- Set operations

Selection

- Syntax: $\sigma_C(R)$
 - R is input
 - C is a condition
- Semantics:
 - Return all tuples that match condition C
 - Set: $\{ t \mid t \in R \text{ AND } t \text{ fulfills } C \}$
 - Bag: $\{ t^i \mid t^i \in R \text{ AND } t \text{ fulfills } C \}$

Selection Example

- $\sigma_{a>5}(R)$

R		Result	
a	b	a	b
1	13		
3	12		
6	14	6	14

Renaming

- Syntax: $\rho_A(R)$
 - R is input
 - A is a list of attribute renamings $b \leftarrow a$
- Semantics:
 - Applies renaming from A to inputs
 - Set: $\{ t.A \mid t \in R \}$
 - Bag: $\{ (t.A)^i \mid t^i \in R \}$

Renaming Example

- $\rho_{c \leftarrow a}(R)$

R		Result	
a	b	c	b
1	13	1	13
3	12	3	12
6	14	6	14

Projection

- Syntax: $\Pi_A(R)$
 - R is input
 - A is a list of projection expressions
 - Standard: only attributes in A
- Semantics:
 - Project all inputs on projection expressions
 - Set: $\{ t.A \mid t \in R \}$
 - Bag: $\{ (t.A)^i \mid t^i \in R \}$

Projection Example

- $\Pi_b(R)$

R		Result
a	b	b
1	13	13
3	12	12
6	14	14

Cross Product

- Syntax: $R \times S$
 - R and S are inputs
- Semantics:
 - All combinations of tuples from R and S
 - = mathematical definition of cross product
 - Set: $\{ (t,s) \mid t \in R \text{ AND } s \in S \}$
 - Bag: $\{ (t,s)^{n^*m} \mid t^i \in R \text{ AND } s^m \in S \}$

Cross Product Example

- $R \times S$

R		S		Result			
a	b	c	d	a	b	c	d
1	13	a	5	1	13	a	5
		b	3	1	13	b	3
		c	4	1	13	c	4
3	12			3	12	a	5
				3	12	b	3
				3	12	c	4

Join

- Syntax: $R \bowtie_C S$
 - R and S are inputs
 - C is a condition
- Semantics:
 - All combinations of tuples from R and S that match C
 - Set: $\{ (t,s) \mid t \in R \text{ AND } s \in S \text{ AND } (t,s) \text{ matches } C \}$
 - Bag: $\{ (t,s)^{n^*m} \mid t^i \in R \text{ AND } s^m \in S \text{ AND } (t,s) \text{ matches } C \}$

Join Example

- $R \bowtie_{a=d} S$

R		S		Result			
a	b	c	d	a	b	c	d
1	13	a	5				
		b	3				
		c	4				
3	12			3	12	b	3

Natural Join

- Syntax: $R \bowtie S$
 - R and S are inputs
- Semantics:
 - All combinations of tuples from R and S that match on common attributes
 - A = common attributes of R and S
 - C = exclusive attributes of S
 - Set: $\{ (t,s,C) \mid t \in R \text{ AND } s \in S \text{ AND } t.A=s.A \}$
 - Bag: $\{ (t,s,C)^{n^*m} \mid t^i \in R \text{ AND } s^m \in S \text{ AND } t.A=s.A \}$

Natural Join Example

- $R \bowtie S$

R		S		Result		
a	b	c	a	a	b	c
1	13	a	5			
		b	3			
		c	4			
3	12			3	12	b

Left-outer Join

- Syntax: $R \bowtie_C S$
 - R and S are inputs
 - C is condition
- Semantics:
 - R join S
 - $t \in R$ without match, fill S attributes with NULL
 - $\{ (t,s) \mid t \in R \text{ AND } s \in S \text{ AND } (t,s) \text{ matches } C \}$
 - union
 - $\{ (t, \text{NULL}(S)) \mid t \in R \text{ AND NOT exists } s \in S: (t,s) \text{ matches } C \}$

Left-outer Join Example

- $R \bowtie_{a=d} S$

R		S		Result			
a	b	c	d	a	b	c	d
1	13	a	5	1	13	NULL	NULL
3	12	b	3	3	12	b	3
		c	4				

Right-outer Join

- Syntax: $R \bowtie_C S$
 - R and S are inputs
 - C is condition
- Semantics:
 - R join S
 - $s \in S$ without match, fill R attributes with NULL
 - $\{ (t,s) \mid t \in R \text{ AND } s \in S \text{ AND } (t,s) \text{ matches } C \}$
 - union
 - $\{ (\text{NULL}(R),s) \mid s \in S \text{ AND NOT exists } t \in R: (t,s) \text{ matches } C \}$

Right-outer Join Example

- $R \bowtie_{a=d} S$

R		S		Result			
a	b	c	d	a	b	c	d
1	13	a	5	NULL	NULL	a	5
3	12	b	3	3	12	b	3
		c	4	NULL	NULL	c	4

Full-outer Join

- Syntax: $R \bowtie_C S$
 - R and S are inputs and C is condition
- Semantics:
 - $\{ (t,s) \mid t \in R \text{ AND } s \in S \text{ AND } (t,s) \text{ matches } C \}$
 - union
 - $\{ (\text{NULL}(R),s) \mid s \in S \text{ AND NOT exists } t \in R: (t,s) \text{ matches } C \}$
 - union
 - $\{ (t, \text{NULL}(S)) \mid t \in R \text{ AND NOT exists } s \in S: (t,s) \text{ matches } C \}$

Full-outer Join Example

- $R \bowtie_{a=d} S$

R		S		Result			
a	b	c	d	a	b	c	d
1	13	a	5	1	13	NULL	NULL
3	12	b	3	NULL	NULL	a	5
		c	4	3	12	b	3
				NULL	NULL	c	4

Semijoin

- Syntax: $R \bowtie S$ and $R \ltimes S$
 - R and S are inputs
- Semantics:
 - All tuples from R that have a matching tuple from relation S on the common attributes A
 - $\{ t \mid t \in R \text{ AND exists } s \in S: t.A = s.A \}$

Semijoin Example

- $R \ltimes S$

R		S		Result	
a	b	c	a	a	b
1	13	a	5	3	12
3	12	b	3		
		c	4		

CS 525



Notes 8 - Parsing and Analysis

85



CS 525



Notes 8 - Parsing and Analysis

86



Antijoin

- Syntax: $R \not\bowtie S$
 - R and S are inputs
- Semantics:
 - All tuples from R that have no matching tuple from relation S on the common attributes A
 - $\{ t \mid t \in R \text{ AND NOT exists } s \in S: t.A = s.A \}$

Antijoin Example

- $R \not\bowtie S$

R		S		Result	
a	b	c	a	a	b
1	13	a	5	1	13
3	12	b	3		
		c	4		

CS 525



Notes 8 - Parsing and Analysis

87



CS 525



Notes 8 - Parsing and Analysis

88



Aggregation

- Syntax: $\sigma_A^G(R)$
 - A is list of aggregation functions
 - G is list of group by attributes
- Semantics:
 - Build groups of tuples according G and compute the aggregation functions from each group
 - $\{ (t.G, \text{agg}(G(t)) \mid t \in R \}$
 - $G(t) = \{ t' \mid t' \in R \text{ AND } t'.G = t.G \}$

Aggregation Example

- $\sigma_{\text{sum}(a)}^b(R)$

R		Result	
a	b	sum(a)	b
1	1	4	1
3	1	9	2
6	2		
3	2		

CS 525



Notes 8 - Parsing and Analysis

89



CS 525



Notes 8 - Parsing and Analysis

90



Duplicate Removal

- Syntax: $\delta(R)$
 - R is input
- Semantics:
 - Remove duplicates from input
 - Set: N/A
 - Bag: $\{ t^1 \mid t^i \in R \}$

Duplicate Removal Example

- $\delta(R)$

R		Result	
a	b	a	b
1	13	1	13
1	13		
6	14	6	14

Set operations

- Input: R and S
 - Have to have the same schema
 - Union compatible
 - Modulo attribute names
- Types
 - Union
 - Intersection
 - Set difference

Union

- Syntax: $R \cup S$
 - R and S are union-compatible inputs
- Semantics:
 - Set: $\{ (t) \mid t \in R \text{ OR } t \in S \}$
 - Bag: $\{ (t,s)^{n+m} \mid t^i \in R \text{ AND } s^m \in S \}$
 - Assumption t^i with $n < 1$ for tuple not in relation

Union Example

- $R \cup S$

R		S		Result	
a		b		a	
1		1		1	
3		2		2	
		3		3	
				1	
				3	

Intersection

- Syntax: $R \cap S$
 - R and S are union-compatible inputs
- Semantics:
 - Set: $\{ (t) \mid t \in R \text{ AND } t \in S \}$
 - Bag: $\{ (t,s)^{\min(n,m)} \mid t^i \in R \text{ AND } s^m \in S \}$

Intersection Example

- $R \cap S$

R	S	Result
a	b	a
1	1	1
3	2	3
	3	

CS 525



Notes 8 - Parsing and Analysis

97



Set Difference

- Syntax: $R - S$
 - R and S are union-compatible inputs
- Semantics:
 - Set: $\{ (t) \mid t \in R \text{ AND NOT } t \in S \}$
 - Bag: $\{ (t,s)^{n-m} \mid t^n \in R \text{ AND } s^m \in S \}$

CS 525



Notes 8 - Parsing and Analysis

98



Set Difference Example

- $R - S$

R	S	Result
a	b	a
1	1	
5	2	5
	3	

CS 525



Notes 8 - Parsing and Analysis

99



Canonical Translation to Relational Algebra

- TEXTBOOK version of conversion
- Given an SQL query
- Return an equivalent relational algebra expression

CS 525



Notes 8 - Parsing and Analysis

100



Canonical Translation

- **FROM** clause into joins and cross-products
 - Cross-product between list items
 - Joins into their algebra counter-part
- **WHERE** clause into selection
- **SELECT** clause into projection and renaming
 - If it has aggregation functions use aggregation
 - **DISTINCT** into duplicate removal

CS 525



Notes 8 - Parsing and Analysis

101



Canonical Translation

- **GROUP BY** clause into aggregation
- **HAVING** clause into selection
- **ORDER BY** - no counter-part
- Then turn joins into crossproducts and selections

CS 525



Notes 8 - Parsing and Analysis

102

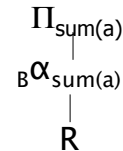


Set Operations

- **UNION ALL** into union
- **UNION** duplicate removal over union
- **INTERSECT ALL** into intersection
- **INTERSECT** add duplicate removal
- **EXCEPT ALL** into set difference
- **EXCEPT** apply duplicate removal to inputs and then apply set difference

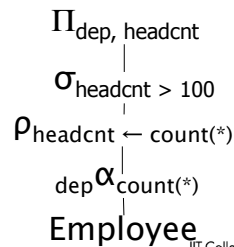
Example: Relational Algebra Translation

```
SELECT sum(R.a)
FROM R
GROUP BY b
```



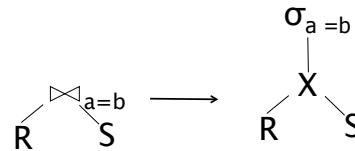
Example: Relational Algebra Translation

```
SELECT dep, headcnt
FROM (SELECT count(*) AS headcnt, dep
      FROM employee
      GROUP BY dep)
WHERE headcnt > 100
```



Example: Relational Algebra Translation

```
SELECT *
FROM R JOIN S ON (R.a = S.b)
```



Parsing and Analysis Summary

- SQL text -> Internal representation
- Semantic checks
- Database catalog
- View unfolding

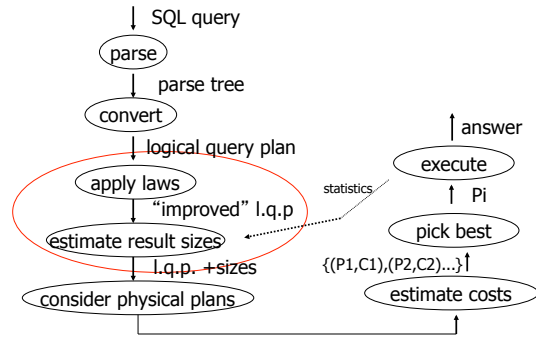


CS 525: Advanced Database Organisation

09: Query Optimization - Logical

Boris Glavic

Slides: adapted from a [course](#) taught by [Hector Garcia-Molina](#), Stanford InfoLab



Query Optimization

- Relational algebra level
- Detailed query plan level

Query Optimization

- Relational algebra level
- Detailed query plan level
 - Estimate Costs
 - without indexes
 - with indexes
 - Generate and compare plans

Relational algebra optimization

- Transformation rules (preserve equivalence)
- What are good transformations?
 - Heuristic application of transformations

Query Equivalence

- Two queries q and q' are equivalent:
 - If for every database instance I
 - Contents of all the tables
 - Both queries have the same result

$$q \equiv q' \text{ iff } \forall I: q(I) = q'(I)$$

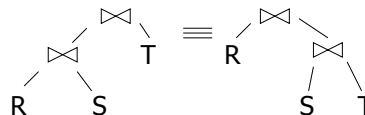
Rules: Natural joins & cross products & union

$$R \bowtie S = S \bowtie R$$

$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

Note:

- Carry attribute names in results, so order is not important
- Can also write as trees, e.g.:



Rules: Natural joins & cross products & union

$$R \bowtie S = S \bowtie R$$

$$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$$

$$R \times S = S \times R$$

$$(R \times S) \times T = R \times (S \times T)$$

$$R \cup S = S \cup R$$

$$R \cup (S \cup T) = (R \cup S) \cup T$$

Rules: Selects

$$\sigma_{p_1 \wedge p_2}(R) =$$

$$\sigma_{p_1 \vee p_2}(R) =$$

Rules: Selects

$$\sigma_{p_1 \wedge p_2}(R) = \sigma_{p_1} [\sigma_{p_2}(R)]$$

$$\sigma_{p_1 \vee p_2}(R) = [\sigma_{p_1}(R)] \cup [\sigma_{p_2}(R)]$$

Bags vs. Sets

$$R = \{a, a, b, b, c\}$$

$$S = \{b, b, c, c, d\}$$

$$R \cup S = ?$$

Bags vs. Sets

R = {a,a,b,b,b,c}

S = {b,b,c,c,d}

RUS = ?

- Option 1 SUM
RUS = {a,a,b,b,b,b,c,c,c,d}
- Option 2 MAX
RUS = {a,a,b,b,b,c,c,d}

Option 2 (MAX) makes this rule work:

$$\sigma_{p1 \vee p2}(R) = \sigma_{p1}(R) \cup \sigma_{p2}(R)$$

Example: R={a,a,b,b,b,c}

P1 satisfied by a,b; P2 satisfied by b,c

Option 2 (MAX) makes this rule work:

$$\sigma_{p1 \vee p2}(R) = \sigma_{p1}(R) \cup \sigma_{p2}(R)$$

Example: R={a,a,b,b,b,c}

P1 satisfied by a,b; P2 satisfied by b,c

$$\sigma_{p1 \vee p2}(R) = \{a,a,b,b,b,c\}$$

$$\sigma_{p1}(R) = \{a,a,b,b,b\}$$

$$\sigma_{p2}(R) = \{b,b,b,c\}$$

$$\sigma_{p1}(R) \cup \sigma_{p2}(R) = \{a,a,b,b,b,c\}$$

“Sum” option makes more sense:

Senators (.....)

Rep (.....)

T1 = $\pi_{yr,state}$ Senators; T2 = $\pi_{yr,state}$ Reps

T1	Yr	State	T2	Yr	State
	97	CA		99	CA
	99	CA		99	CA
	98	AZ		98	CA

Union?

Executive Decision

- > Use “SUM” option for bag unions
- > Some rules cannot be used for bags

Rules: Project

- Let: X = set of attributes
- Y = set of attributes
- XY = X U Y

$$\pi_{xy}(R) =$$

Rules: Project

Let: X = set of attributes
 Y = set of attributes
 XY = X U Y

$$\pi_{xy}(R) = \pi_x[\pi_y(R)]$$

Rules: Project

Let: X = set of attributes
 Y = set of attributes
 XY = X U Y

~~$$\pi_{xy}(R) = \pi_x[\pi_y(R)]$$~~

Rules: $\sigma + \bowtie$ combined

Let p = predicate with only R attribs
 q = predicate with only S attribs
 m = predicate with only R,S attribs

$$\sigma_p(R \bowtie S) =$$

$$\sigma_q(R \bowtie S) =$$

Rules: $\sigma + \bowtie$ combined

Let p = predicate with only R attribs
 q = predicate with only S attribs
 m = predicate with only R,S attribs

$$\sigma_p(R \bowtie S) = [\sigma_p(R)] \bowtie S$$

$$\sigma_q(R \bowtie S) = R \bowtie [\sigma_q(S)]$$

Rules: $\sigma + \bowtie$ combined (continued)

Some Rules can be Derived:

$$\sigma_{p \wedge q}(R \bowtie S) =$$

$$\sigma_{p \wedge q \wedge m}(R \bowtie S) =$$

$$\sigma_{p \vee q}(R \bowtie S) =$$

Do one:

$$\sigma_{p \wedge q}(R \bowtie S) = [\sigma_p(R)] \bowtie [\sigma_q(S)]$$

$$\sigma_{p \wedge q \wedge m}(R \bowtie S) = \sigma_m[(\sigma_p R) \bowtie (\sigma_q S)]$$

$$\sigma_{p \vee q}(R \bowtie S) = [(\sigma_p R) \bowtie S] \cup [R \bowtie (\sigma_q S)]$$

--> Derivation for first one:

$$\sigma_{p \wedge q} (R \bowtie S) =$$

$$\sigma_p [\sigma_q (R \bowtie S)] =$$

$$\sigma_p [R \bowtie \sigma_q (S)] =$$

$$[\sigma_p (R)] \bowtie [\sigma_q (S)]$$

CS 525



Notes 9 - Logical Optimization

25



Rules: π, σ combined

Let x = subset of R attributes

z = attributes in predicate P
(subset of R attributes)

$$\pi_x [\sigma_p (R)] =$$

CS 525



Notes 9 - Logical Optimization

26



Rules: π, σ combined

Let x = subset of R attributes

z = attributes in predicate P
(subset of R attributes)

$$\pi_x [\sigma_p (R)] = \{ \sigma_p [\pi_x (R)] \}$$

CS 525



Notes 9 - Logical Optimization

27



Rules: π, σ combined

Let x = subset of R attributes

z = attributes in predicate P
(subset of R attributes)

$$\pi_x [\sigma_p (R)] = \pi_x \{ \sigma_p [\pi_{xz} (R)] \}$$

CS 525



Notes 9 - Logical Optimization

28



Rules: π, \bowtie combined

Let x = subset of R attributes

y = subset of S attributes

z = intersection of R,S attributes

$$\pi_{xy} (R \bowtie S) =$$

CS 525



Notes 9 - Logical Optimization

29



Rules: π, \bowtie combined

Let x = subset of R attributes

y = subset of S attributes

z = intersection of R,S attributes

$$\pi_{xy} (R \bowtie S) =$$

$$\pi_{xy} \{ [\pi_{xz} (R)] \bowtie [\pi_{yz} (S)] \}$$

CS 525



Notes 9 - Logical Optimization

30



$$\pi_{xy} \{ \sigma_p (R \bowtie S) \} =$$

$$\pi_{xy} \{ \sigma_p (R \bowtie S) \} =$$

$$\pi_{xy} \{ \sigma_p [\pi_{xz'} (R) \bowtie \pi_{yz'} (S)] \}$$

$$z' = z \cup \{ \text{attributes used in P} \}$$

Rules for σ, π combined with X

similar...

e.g., $\sigma_p (R \times S) = ?$

Rules σ, \cup combined:

$$\sigma_p (R \cup S) = \sigma_p (R) \cup \sigma_p (S)$$

$$\sigma_p (R - S) = \sigma_p (R) - S = \sigma_p (R) - \sigma_p (S)$$

Which are “good” transformations?

- $\sigma_{p_1 \wedge p_2} (R) \rightarrow \sigma_{p_1} [\sigma_{p_2} (R)]$
- $\sigma_p (R \bowtie S) \rightarrow [\sigma_p (R)] \bowtie S$
- $R \bowtie S \rightarrow S \bowtie R$
- $\pi_x [\sigma_p (R)] \rightarrow \pi_x \{ \sigma_p [\pi_{xz} (R)] \}$

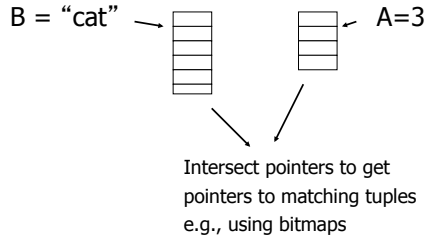
Conventional wisdom:

do projects early

Example: $R(A,B,C,D,E) \quad x=\{E\}$
 $P: (A=3) \wedge (B=\text{“cat”})$

$$\pi_x \{ \sigma_p (R) \} \quad \text{vs.} \quad \pi_E \{ \sigma_p \{ \pi_{ABE} (R) \} \}$$

But What if we have A, B indexes?



CS 525



Notes 9 - Logical Optimization

37



Bottom line:

- No transformation is always good
- Usually good: early selections
 - Exception: expensive selection conditions
 - E.g., UDFs

CS 525



Notes 9 - Logical Optimization

38



More transformations

- Eliminate common sub-expressions
- Detect constant expressions
- Other operations: duplicate elimination

CS 525



Notes 9 - Logical Optimization

39



Pushing Selections

- Idea:
 - Join conditions equate attributes
 - For parts of algebra tree (scope) store which attributes have to be the same
 - Called Equivalence classes
- Example: $R(a,b), S(c,d)$

$$\sigma_{b=3} (R \bowtie_{b=c} S) = \sigma_{b=3} (R) \bowtie_{b=c} \sigma_{c=3} (S)$$

CS 525



Notes 9 - Logical Optimization

40



Outer-Joins

- Not commutative
 - $R \bowtie S \neq S \bowtie R$
 - p – condition over attributes in A
 - A list of attributes from R
- $$\sigma_p (R \bowtie_{A=B} S) \equiv \sigma_p (R) \bowtie_{A=B} S$$
- Not $\sigma_p (R \bowtie_{A=B} S) \equiv R \bowtie_{A=B} \sigma_p (S)$

CS 525



Notes 9 - Logical Optimization

41



Summary Equivalences

- Associativity: $(R \circ S) \circ T \equiv R \circ (S \circ T)$
- Commutativity: $R \circ S \equiv S \circ R$
- Distributivity: $(R \circ S) \otimes T \equiv (R \otimes T) \circ (S \otimes T)$
- Difference between Set and Bag Equivalences
- Only some equivalence are useful

CS 525



Notes 9 - Logical Optimization

42



Outline - Query Processing

- Relational algebra level
 - transformations
 - good transformations
- Detailed query plan level
 - estimate costs
 - generate and compare plans

- Estimating cost of query plan

- (1) Estimating size of results
- (2) Estimating # of IOs

Estimating result size

- Keep statistics for relation R
 - $T(R)$: # tuples in R
 - $S(R)$: # of bytes in each R tuple
 - $B(R)$: # of blocks to hold all R tuples
 - $V(R, A)$: # distinct values in R for attribute A

Example

R	A	B	C	D
cat	1	10	a	
cat	1	20	b	
dog	1	30	a	
dog	1	40	c	
bat	1	50	d	

A: 20 byte string
 B: 4 byte integer
 C: 8 byte date
 D: 5 byte string

Example

R	A	B	C	D
cat	1	10	a	
cat	1	20	b	
dog	1	30	a	
dog	1	40	c	
bat	1	50	d	

A: 20 byte string
 B: 4 byte integer
 C: 8 byte date
 D: 5 byte string

$$T(R) = 5 \quad S(R) = 37$$

$$V(R,A) = 3 \quad V(R,C) = 5$$

$$V(R,B) = 1 \quad V(R,D) = 4$$

Size estimates for $W = R_1 \times R_2$

$$T(W) =$$

$$S(W) =$$

Size estimates for $W = R_1 \times R_2$

$$T(W) = T(R_1) \times T(R_2)$$

$$S(W) = S(R_1) + S(R_2)$$

Size estimate for $W = \sigma_{A=a}(R)$

$$S(W) = S(R)$$

$$T(W) = ?$$

Example

R	A	B	C	D
cat	1	10	a	
cat	1	20	b	
dog	1	30	a	
dog	1	40	c	
bat	1	50	d	

$$V(R,A)=3$$

$$V(R,B)=1$$

$$V(R,C)=5$$

$$V(R,D)=4$$

Example

R	A	B	C	D
cat	1	10	a	
cat	1	20	b	
dog	1	30	a	
dog	1	40	c	
bat	1	50	d	

$$V(R,A)=3$$

$$V(R,B)=1$$

$$V(R,C)=5$$

$$V(R,D)=4$$

$$W = \sigma_{Z=val}(R) \quad T(W) =$$

$$W = \sigma_{Z=val}(R) \quad T(W) = \frac{T(R)}{V(R,Z)}$$

Assumption:

Values in select expression $Z = val$ are uniformly distributed over possible $V(R,Z)$ values.

Alternate Assumption:

Values in select expression $Z = val$ are uniformly distributed over domain with $DOM(R,Z)$ values.

Example

R	A	B	C	D
cat	1	10	a	
cat	1	20	b	
dog	1	30	a	
dog	1	40	c	
bat	1	50	d	

Alternate assumption
 $V(R,A)=3 \quad \text{DOM}(R,A)=10$
 $V(R,B)=1 \quad \text{DOM}(R,B)=10$
 $V(R,C)=5 \quad \text{DOM}(R,C)=10$
 $V(R,D)=4 \quad \text{DOM}(R,D)=10$

$$C=\text{val} \Rightarrow T(W) = (1/10)1 + (1/10)1 + \dots = (5/10) = 0.5$$

$$B=\text{val} \Rightarrow T(W) = (1/10)5 + 0 + 0 = 0.5$$

$$A=\text{val} \Rightarrow T(W) = (1/10)2 + (1/10)2 + (1/10)1 = 0.5$$

$$W = \sigma_{z=\text{val}}(R) \quad T(W) = ?$$

CS 525



Notes 9 - Logical Optimization

55



CS 525



Notes 9 - Logical Optimization

56



Example

R	A	B	C	D
cat	1	10	a	
cat	1	20	b	
dog	1	30	a	
dog	1	40	c	
bat	1	50	d	

Alternate assumption
 $V(R,A)=3 \quad \text{DOM}(R,A)=10$
 $V(R,B)=1 \quad \text{DOM}(R,B)=10$
 $V(R,C)=5 \quad \text{DOM}(R,C)=10$
 $V(R,D)=4 \quad \text{DOM}(R,D)=10$

Selection cardinality

$SC(R,A)$ = average # records that satisfy equality condition on R.A

$$SC(R,A) = \begin{cases} \frac{T(R)}{V(R,A)} \\ \frac{T(R)}{\text{DOM}(R,A)} \end{cases}$$

$$W = \sigma_{z=\text{val}}(R) \quad T(W) = \frac{T(R)}{\text{DOM}(R,Z)}$$

CS 525



Notes 9 - Logical Optimization

57



CS 525



Notes 9 - Logical Optimization

58



What about $W = \sigma_{z \geq \text{val}}(R)$?

$T(W) = ?$

What about $W = \sigma_{z \geq \text{val}}(R)$?

$T(W) = ?$

- Solution # 1:

$$T(W) = T(R)/2$$

CS 525



Notes 9 - Logical Optimization

59



CS 525



Notes 9 - Logical Optimization

60



What about $W = \sigma_{z \geq \text{val}}(R)$?

$T(W) = ?$

- Solution # 1:

$$T(W) = T(R)/2$$

- Solution # 2:

$$T(W) = T(R)/3$$

CS 525



Notes 9 - Logical Optimization

61

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

- Solution # 3: Estimate values in range

Example R

	Z

Min=1
↑
Max=20

$V(R,Z)=10$

$W = \sigma_{z \geq 15}(R)$

CS 525



Notes 9 - Logical Optimization

62

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

- Solution # 3: Estimate values in range

Example R

	Z

Min=1
↑
Max=20

$V(R,Z)=10$
 $W = \sigma_{z \geq 15}(R)$

$$f = \frac{20-15+1}{20-1+1} = \frac{6}{20} \quad (\text{fraction of range})$$

$$T(W) = f \times T(R)$$

Equivalently:

$f \times V(R,Z) = \text{fraction of distinct values}$

$$T(W) = \left[\frac{f \times V(Z,R)}{V(Z,R)} \right] \times T(R) = f \times T(R)$$

CS 525



Notes 9 - Logical Optimization

63

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

CS 525



Notes 9 - Logical Optimization

64

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Size estimate for $W = R1 \bowtie R2$

Let x = attributes of $R1$

y = attributes of $R2$

Size estimate for $W = R1 \bowtie R2$

Let x = attributes of $R1$

y = attributes of $R2$

Case 1

$$X \cap Y = \emptyset$$

Same as $R1 \times R2$

CS 525



Notes 9 - Logical Optimization

65

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

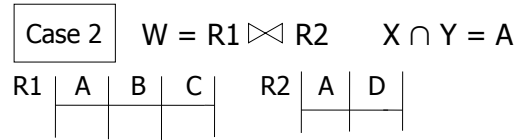
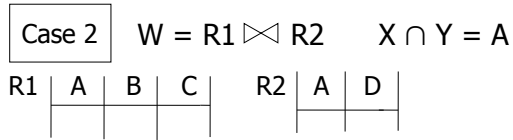
CS 525



Notes 9 - Logical Optimization

66

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY



Assumption:

$V(R1,A) \leq V(R2,A) \Rightarrow$ Every A value in R1 is in R2

$V(R2,A) \leq V(R1,A) \Rightarrow$ Every A value in R2 is in R1

CS 525



Notes 9 - Logical Optimization

67



CS 525

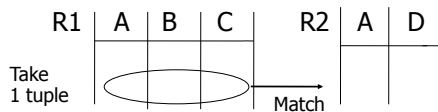


Notes 9 - Logical Optimization

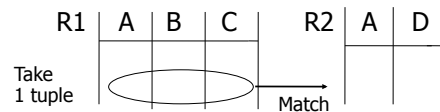
68



Computing T(W) when $V(R1,A) \leq V(R2,A)$



Computing T(W) when $V(R1,A) \leq V(R2,A)$



1 tuple matches with $\frac{T(R2)}{V(R2,A)}$ tuples...

$$\text{so } T(W) = \frac{T(R2)}{V(R2,A)} \times T(R1)$$

CS 525



Notes 9 - Logical Optimization

69



CS 525



Notes 9 - Logical Optimization

70



- $V(R1,A) \leq V(R2,A) \quad T(W) = \frac{T(R2) T(R1)}{V(R2,A)}$

- $V(R2,A) \leq V(R1,A) \quad T(W) = \frac{T(R2) T(R1)}{V(R1,A)}$

In general $W = R1 \bowtie R2$

$$T(W) = \frac{T(R2) T(R1)}{\max\{V(R1,A), V(R2,A)\}}$$

[A is common attribute]

CS 525



Notes 9 - Logical Optimization

71



CS 525



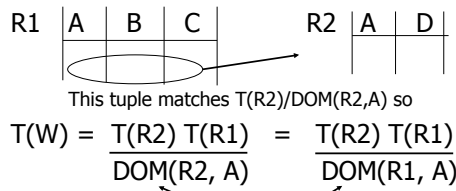
Notes 9 - Logical Optimization

72



Case 2 with alternate assumption

Values uniformly distributed over domain



In all cases:

$$S(W) = S(R1) + S(R2) - S(A)$$

size of attribute A

Using similar ideas,
we can estimate sizes of:

- $\Pi_{AB}(R)$
- $\sigma_{A=a \wedge B=b}(R)$
- $R \bowtie S$ with common attribs. A,B,C
- Union, intersection, diff,

Note: for complex expressions, need intermediate T,S,V results.

E.g. $W = [\sigma_{A=a}(R1)] \bowtie R2$

Treat as relation U

$$T(U) = T(R1)/V(R1,A) \quad S(U) = S(R1)$$

Also need $V(U, *)$!!

To estimate Vs

- E.g., $U = \sigma_{A=a}(R1)$
Say R1 has attribs A,B,C,D
- $V(U, A) =$
 - $V(U, B) =$
 - $V(U, C) =$
 - $V(U, D) =$

Example

R1	A	B	C	D
cat	1	10	10	
cat	1	20	20	
dog	1	30	10	
dog	1	40	30	
bat	1	50	10	

- $V(R1,A)=3$
- $V(R1,B)=1$
- $V(R1,C)=5$
- $V(R1,D)=3$
- $U = \sigma_{A=a}(R1)$

Example

R1	A	B	C	D
cat	1	10	10	
cat	1	20	20	
dog	1	30	10	
dog	1	40	30	
bat	1	50	10	

$$V(R1,A)=3$$

$$V(R1,B)=1$$

$$V(R1,C)=5$$

$$V(R1,D)=3$$

$$U = \sigma_{A=a}(R1)$$

Possible Guess $U = \sigma_{A=a}(R)$

$$V(U,A) = 1$$

$$V(U,B) = V(R,B)$$

$$V(U,A) = 1 \quad V(U,B) = 1 \quad V(U,C) = \frac{T(R1)}{V(R1,A)}$$

$V(D,U)$... somewhere in between

CS 525



Notes 9 - Logical Optimization

79



CS 525



Notes 9 - Logical Optimization

80



For Joins $U = R1(A,B) \bowtie R2(A,C)$

$$V(U,A) = \min \{ V(R1, A), V(R2, A) \}$$

$$V(U,B) = V(R1, B)$$

$$V(U,C) = V(R2, C)$$

Example:

$$Z = R1(A,B) \bowtie R2(B,C) \bowtie R3(C,D)$$

R1	T(R1) = 1000	V(R1,A)=50	V(R1,B)=100
----	--------------	------------	-------------

R2	T(R2) = 2000	V(R2,B)=200	V(R2,C)=300
----	--------------	-------------	-------------

R3	T(R3) = 3000	V(R3,C)=90	V(R3,D)=500
----	--------------	------------	-------------

CS 525



Notes 9 - Logical Optimization

81



CS 525



Notes 9 - Logical Optimization

82



Partial Result: $U = R1 \bowtie R2$

$$T(U) = \frac{1000 \times 2000}{200}$$

$$V(U,A) = 50$$

$$V(U,B) = 100$$

$$V(U,C) = 300$$

$$Z = U \bowtie R3$$

$$T(Z) = \frac{1000 \times 2000 \times 3000}{200 \times 300}$$

$$V(Z,A) = 50$$

$$V(Z,B) = 100$$

$$V(Z,C) = 90$$

$$V(Z,D) = 500$$

CS 525



Notes 9 - Logical Optimization

83



CS 525



Notes 9 - Logical Optimization

84

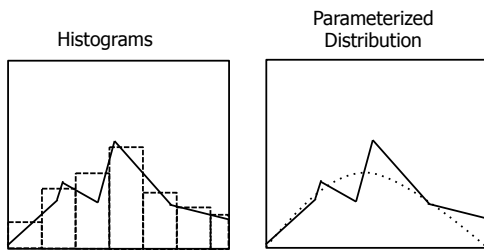


Approximating Distributions

- Summarize the distribution
 - Used to better estimate result sizes
 - Without the need to look at all the data
- Concerns
 - Error metric: How to measure preciseness
 - Memory consumption
 - Computational Complexity

Approximating Distributions

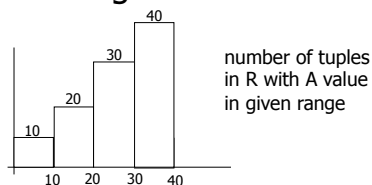
- Parameterized distribution
 - E.g., gauss distribution
 - Adapt parameters to fit data
- Histograms
 - Divide domain into ranges (buckets)
 - Store the number of tuples per bucket
- Both need to be maintained



Maintaining Statistics

- Use separate command that triggers statistics collection
 - Postgres: ANALYZE
- During query processing
 - Overhead for queries
- Use Sampling?

Estimating Result Size using Histograms



$$\sigma_{A=val}(R) = ?$$

Estimating Result Size using Histograms

- $\sigma_{A=val}(R) = ?$
- $|B|$ - number of values per bucket
- $\#B$ - number of records in bucket

$$\frac{\#B}{|B|}$$

Join Size using Histograms

- $R \bowtie S$
- Use

$$T(W) = \frac{T(R2) T(R1)}{\max\{V(R1,A), V(R2,A)\}}$$

- Apply for each bucket

Join Size using Histograms

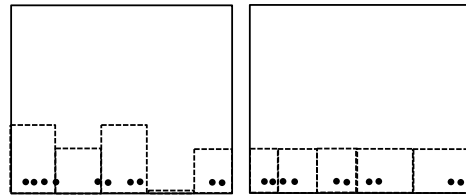
- $V(R1,A) = V(R2,A) = \text{bucket size } |B|$

$$T(W) = \sum_{\text{buckets}} \frac{\#B(R2) \#B(R1)}{|B|}$$

Equi-width vs. Equi-depth

- Equi-width
 - All buckets contain the same number of values
 - Easy, but inaccurate
- Equi-depth (used by most DBMS)
 - All buckets contain the same number of tuples
 - Better accuracy, need to sort data to compute

Equi-width vs. Equi-depth



Construct Equi-depth Histograms

- Sort input
- Determine size of buckets
 - #bucket / #tuples
- Example 3 buckets
 1, 5, 44, 6, 10, 12, 3, 6, 7
 1, 3, 5, 6, 6, 7, 10, 12, 44
 [1-5] [6-8] [9-44]

Advanced Techniques

- Wavelets
- Approximate Histograms
- Sampling Techniques
- Compressed Histograms

Summary

- Estimating size of results is an “art”
- Don't forget:
Statistics must be kept up to date...
(cost?)


Outline

- Estimating cost of query plan
 - Estimating size of results ← done!
 - Estimating # of IOs ← next...
 - Operator Implementations
- Generate and compare plans



CS 525: Advanced Database Organization

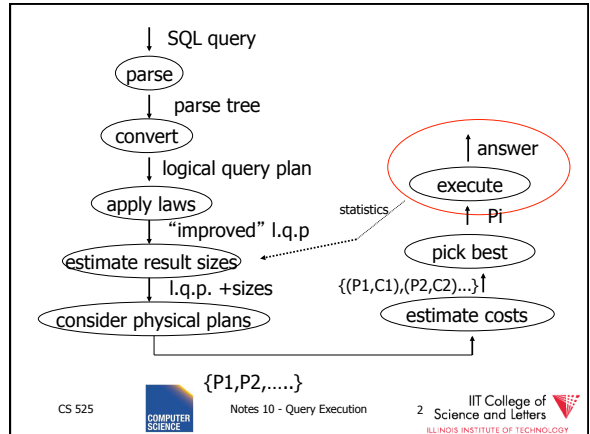
10: Query Execution

Boris Glavic





Slides: adapted from a [course](#) taught by [Hector Garcia-Molina](#), Stanford InfoLab

CS 525  Notes 10 - Query Execution 1 





Query Execution

- Here only:
 - how to implement operators
 - what are the costs of implementations
 - how to implement queries
 - Data flow between operators
- Next part:
 - How to choose good plan

CS 525  Notes 10 - Query Execution 3 



Execution Plan

- A tree (DAG) of physical operators that implement a query
- May use indices
- May create temporary relations
- May create indices on the fly
- May use auxiliary operations such as sorting

CS 525  Notes 10 - Query Execution 4 



How to estimate costs

- If everything fits into memory
 - Standard computational complexity
- If not
 - Assume fixed memory available for buffering pages
 - Count I/O operations
 - Real systems combine this with CPU estimations

CS 525  Notes 10 - Query Execution 5 

Estimating IOs:

- Count # of disk blocks that must be read (or written) to execute query plan

CS 525  Notes 10 - Query Execution 6 

To estimate costs, we may have additional parameters:

$B(R)$ = # of blocks containing R tuples

$f(R)$ = max # of tuples of R per block

M = # memory blocks available

CS 525



Notes 10 - Query Execution

7

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

To estimate costs, we may have additional parameters:

$B(R)$ = # of blocks containing R tuples

$f(R)$ = max # of tuples of R per block

M = # memory blocks available

$HT(i)$ = # levels in index i

$LB(i)$ = # of leaf blocks in index i

CS 525



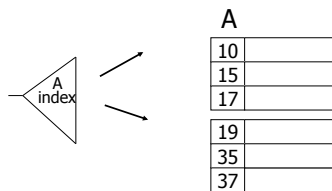
Notes 10 - Query Execution

8

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Clustered index

Index that allows tuples to be read in an order that corresponds to physical order



CS 525



Notes 10 - Query Execution

9

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Operators Overview

- (External) Sorting
- Joins (Nested Loop, Merge, Hash, ...)
- Aggregation (Sorting, Hash)
- Selection, Projection (Index, Scan)
- Union, Set Difference
- Intersection
- Duplicate Elimination

CS 525



Notes 10 - Query Execution

10

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Operator Profiles

- Algorithm
- In-memory complexity: e.g., $O(n^2)$
- Memory requirements
 - Runtime based on available memory
- #I/O if operation needs to go to disk
- Disk space needed
- Prerequisites
 - Conditions under which the operator can be applied

CS 525



Notes 10 - Query Execution

11

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Execution Strategies

- Compiled
 - Translate into C/C++/Assembler code
 - Compile, link, and execute code
- Interpreted
 - Generic operator implementations
 - Generic executor
 - Interprets query plan

CS 525



Notes 10 - Query Execution

12

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Virtual Machine Approach

- Implement virtual machine of low-level DBMS operations
- Compile query into machine-code for that machine

CS 525



Notes 10 - Query Execution

13

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Iterator Model

- Need to be able to combine operators in different ways
 - E.g., join inputs may be scans, or outputs of other joins, ...
 - -> define generic interface for operators
 - be able to arbitrarily compose complex plans from a small set of operators

CS 525



Notes 10 - Query Execution

14

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Iterator Model - Interface

- **Open**
 - Prepare operator to read inputs
- **Close**
 - Close operator and clean up
- **Next**
 - Return next result tuple

CS 525

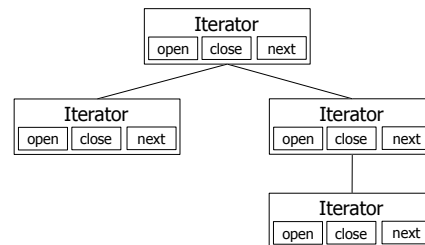


Notes 10 - Query Execution

15

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Query Execution – Iterator Model



CS 525

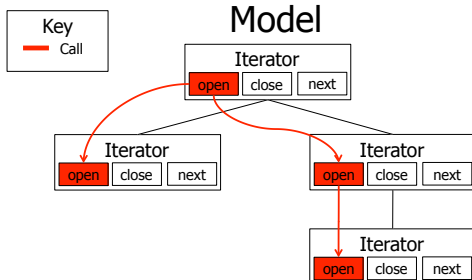


Notes 10 - Query Execution

16

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Query Execution – Iterator Model



CS 525

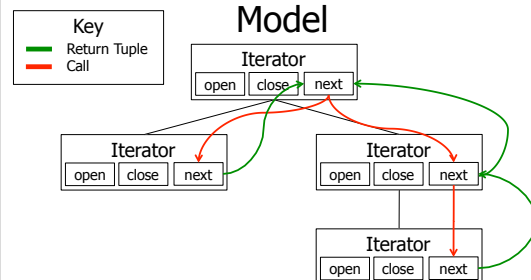


Notes 10 - Query Execution

17

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Query Execution – Iterator Model



CS 525



Notes 10 - Query Execution

18

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Parallelism

- Iterator Model
 - **Pull-based** query execution
- Potential types of parallelism
 - Inter-query (every multiuser system)
 - Intra-operator
 - Inter-operator

CS 525



Notes 10 - Query Execution

19

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Intra-Operator Parallelism

- Execute portions of an operator in parallel
 - Merge-Sort
 - Assign a processor to each merge phase
 - Scan
 - Partition tables
 - Each process scans one partition

CS 525



Notes 10 - Query Execution

20

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Inter-Operator Parallelism

- Each process executes one or more operators
- **Pipelining**
 - **Push-based** query execution
 - Chain operators to directly produce results
 - Pipeline-breakers
 - Operators that need to consume the whole input (or large parts) before producing outputs

CS 525



Notes 10 - Query Execution

21

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Pipelining Communication

- Queues
 - Operators push their results to queues
 - Operators read their inputs from queues
- Direct call
 - Operator calls its parent in the tree with results
 - Within one process

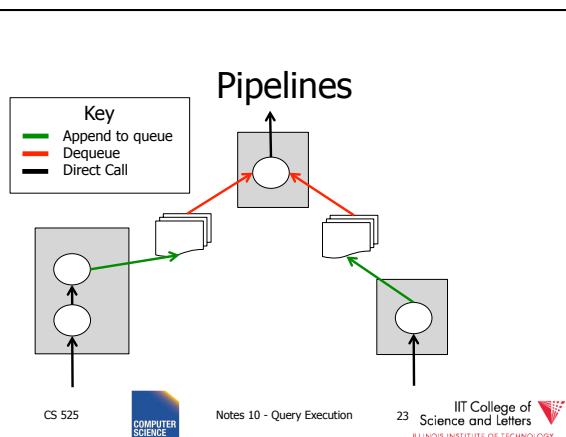
CS 525



Notes 10 - Query Execution

22

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY



Pipeline-breakers

- Sorting
 - All operators that apply sorting
- Aggregation
- Set Difference
- Some implementations of
 - Join
 - Union

CS 525



Notes 10 - Query Execution

24

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Operators Overview

- (External) Sorting
- Joins (Nested Loop, Merge, Hash, ...)
- Aggregation (Sorting, Hash)
- Selection, Projection (Index, Scan)
- Union, Set Difference
- Intersection
- Duplicate Elimination

CS 525



Notes 10 - Query Execution

25

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Sorting

- Why do we want/need to sort
 - Query requires sorting (ORDER BY)
 - Operators require sorted input
 - Merge-join
 - Aggregation by sorting
 - Duplicate removal using sorting

CS 525



Notes 10 - Query Execution

26

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

In-memory sorting

- Algorithms from data structures 101
 - Quick sort
 - Merge sort
 - Heap sort
 - Intro sort
 - ...

CS 525



Notes 10 - Query Execution

27

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

External sorting

- Problem:
 - Sort **N** pages of data with **M** pages of memory
- Solutions?

CS 525



Notes 10 - Query Execution

28

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

First Idea

- Split data into runs of size **M**
- Sort each run in memory and write back to disk
 - $\lceil N/M \rceil$ sorted runs of size **M**
- Now what?



CS 525



Notes 10 - Query Execution

29

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Merging Runs

- Need to create bigger sorted runs out of sorted smaller runs
 - Divide and Conquer
 - Merge Sort?
- How to merge two runs that are bigger than **M**?

CS 525



Notes 10 - Query Execution

30

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Merging Runs using 3 pages

- Merging sorted runs R_1 and R_2
- Need 3 pages
 - One page to buffer pages from R_1
 - One page to buffer pages from R_2
 - One page to buffer the result
 - Whenever this buffer is full, write it to disk

CS 525

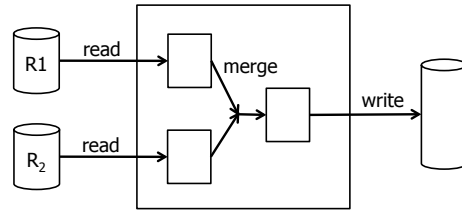


Notes 10 - Query Execution

31

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Merging Runs



CS 525



Notes 10 - Query Execution

32

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

2-Way External Mergesort

- Repeat process until we have one sorted run
- Each iteration (pass) reads and writes the whole table once: $2 B(R)$ I/Os
- Each pass doubles the run size
 - $1 + \lceil \log_2 (B(R) / M) \rceil$ runs
 - $2 B(R) * (1 + \lceil \log_2 (B(R) / M) \rceil)$ I/Os

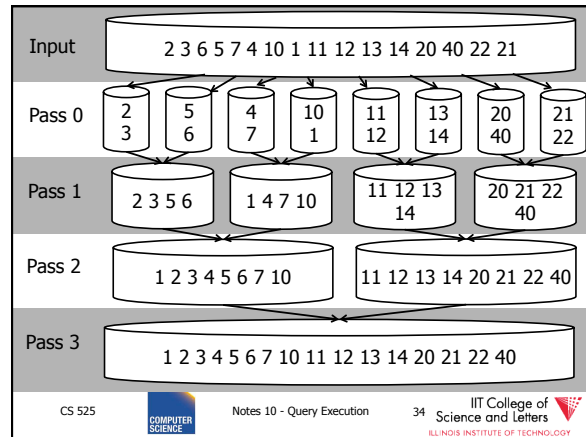
CS 525



Notes 10 - Query Execution

33

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY



CS 525



Notes 10 - Query Execution

34

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

N-Way External Mergesort

- How to utilize M buffer during merging?
- Each pass merges $M-1$ runs at once
 - One memory page as buffer for each run
- #I/Os
 - $1 + \lceil \log_{M-1} (B(R) / M) \rceil$ runs
 - $2 B(R) * (1 + \lceil \log_{M-1} (B(R) / M) \rceil)$ I/Os

CS 525

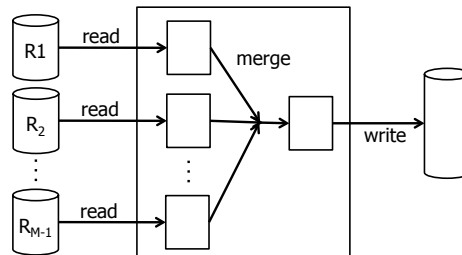


Notes 10 - Query Execution

35

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Merging Runs



CS 525



Notes 10 - Query Execution

36

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

How many passes do we need?

N	M=17	M=129	M=257	M=513	M=1025
100	2	1	1	1	1
1,000	3	2	2	2	1
10,000	4	2	2	2	2
100,000	5	3	3	2	2
1,000,000	5	3	3	3	2
10,000,000	6	4	3	3	3
100,000,000	7	4	4	3	3
1,000,000,000	8	5	4	4	3

CS 525



Notes 10 - Query Execution

37

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

To put into perspective

- Scenario
 - Page size 4KB
 - 1TB of data (250,000,000)
 - 10MB of buffer for sorting (250)
- Passes
 - 4 passes

CS 525



Notes 10 - Query Execution

38

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Merge

- In practice would want larger I/O buffer for each run
- Trade-off between number of runs and efficiency of I/O

CS 525



Notes 10 - Query Execution

39

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Improving in-memory merging

- Merging **M** runs
 - To choose next element to output
 - Have to compare **M** elements
 - \rightarrow complexity linear in **M**: **O(M)**
- How to improve that?
 - Use priority queue to store current element from each run
 - \rightarrow **O(log₂(M))**

CS 525



Notes 10 - Query Execution

40

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Priority Queue

- Queue for accessing elements in some given order
 - **pop-smallest** = return and remove smallest element in set
 - **Insert(e)** = insert element into queue

CS 525



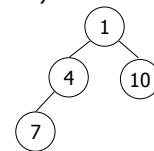
Notes 10 - Query Execution

41

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Min-Heap

- Implementation of priority queue
 - Store elements in a binary tree
 - All levels are full (except leaf level)
 - Heap property
 - Parent is smaller than child
- Example: { 1, 4, 7, 10 }



CS 525



Notes 10 - Query Execution

42

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Min-Heap Insertion

- **insert(e)**

1. Add element at next free leaf node
 - This may invalidate heap property
2. If node smaller than parent then
 - Switch node with parent
3. Repeat until 2) cannot be applied anymore

CS 525



Notes 10 - Query Execution

43

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Min-Heap Dequeue

- **pop-smallest**

1. Return Root and use right-most leaf as new root
 - This may invalidate heap property
2. If node smaller than child then
 - Switch node with smaller child
3. Repeat until 2) cannot be applied anymore

CS 525



Notes 10 - Query Execution

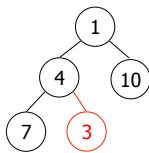
44

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

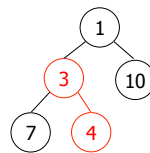
Insertion

- **Insert 3**

Insert at first free position



Restore heap property



CS 525

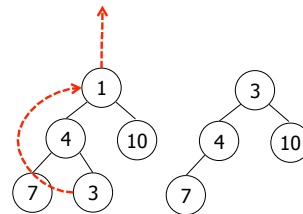


Notes 10 - Query Execution

45

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Dequeue



CS 525

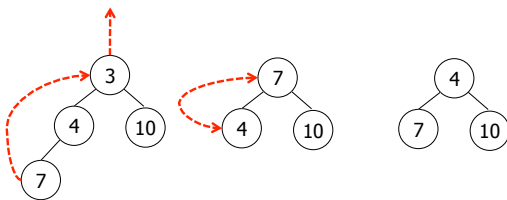


Notes 10 - Query Execution

46

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Dequeue



CS 525



Notes 10 - Query Execution

47

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Min/Max-Heap Complexity

- Heap is a complete tree
 - Height is $O(\log_2(n))$
- Insertion
 - Maximal height of the tree switches
 - $\rightarrow O(\log_2(n))$
- Dequeue
 - Maximal height of the tree switches
 - $\rightarrow O(\log_2(n))$

CS 525



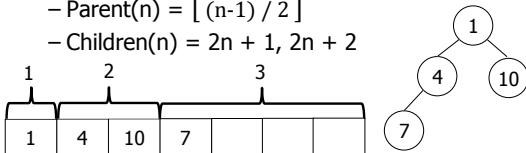
Notes 10 - Query Execution

48

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Min-Heap Implementation

- Full tree
 - Use array to implement tree
- Compute positions
 - $\text{Parent}(n) = \lfloor (n-1) / 2 \rfloor$
 - $\text{Children}(n) = 2n + 1, 2n + 2$



CS 525

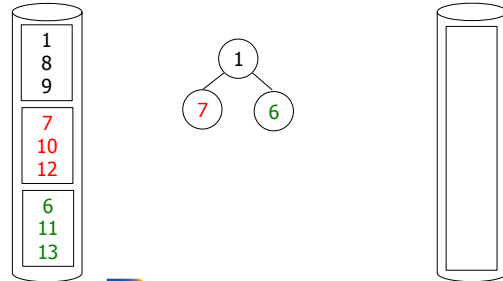


Notes 10 - Query Execution

49

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Merging with Priority Queue



CS 525

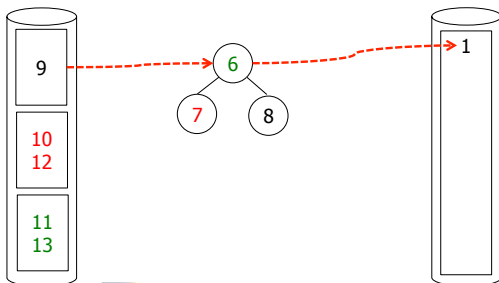


Notes 10 - Query Execution

50

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Merging with Priority Queue



CS 525

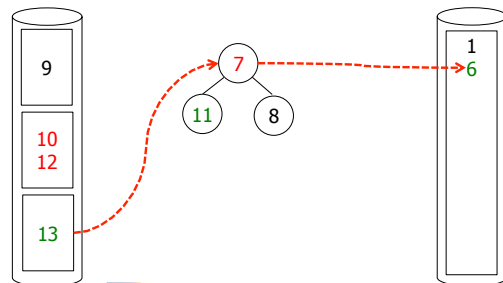


Notes 10 - Query Execution

51

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Merging with Priority Queue



CS 525



Notes 10 - Query Execution

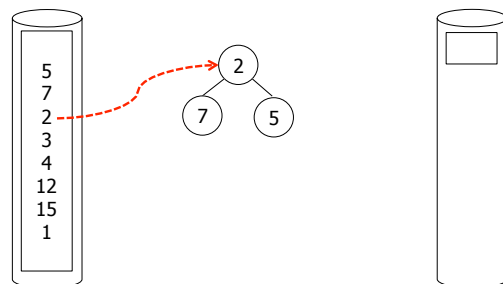
52

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Using a heap to generate runs

- Read inputs into heap
 - Until available memory is full
- Replace elements
 - Remove smallest element from heap
 - If larger then last element written of current run then write to current run
 - Else create a new run
 - Add new element from input to heap

Using a heap to generate runs



CS 525



Notes 10 - Query Execution

53

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

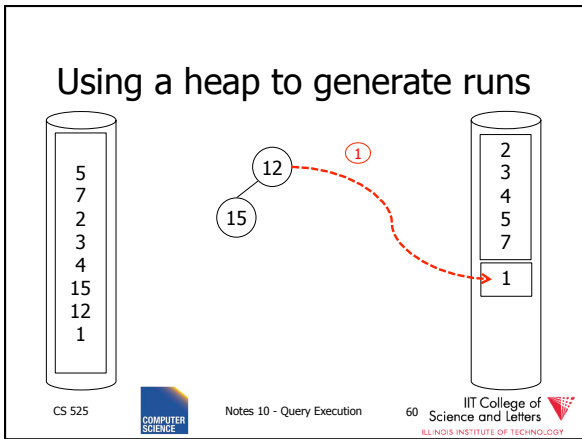
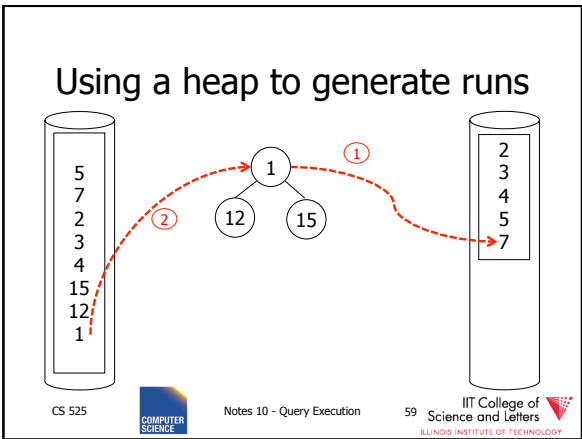
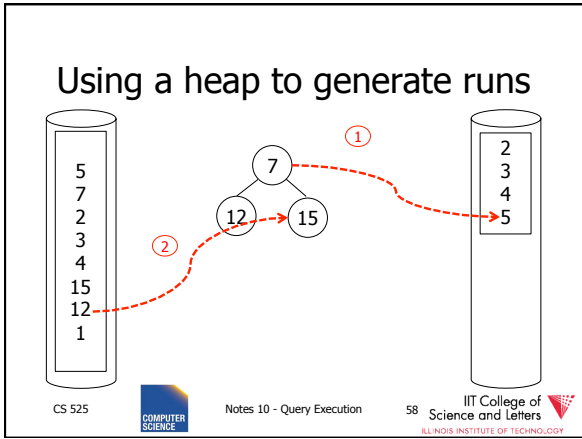
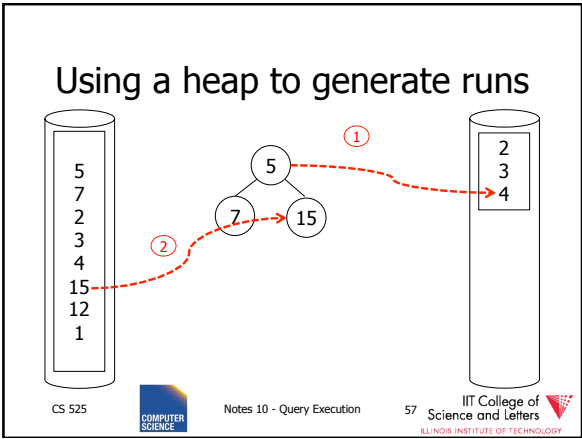
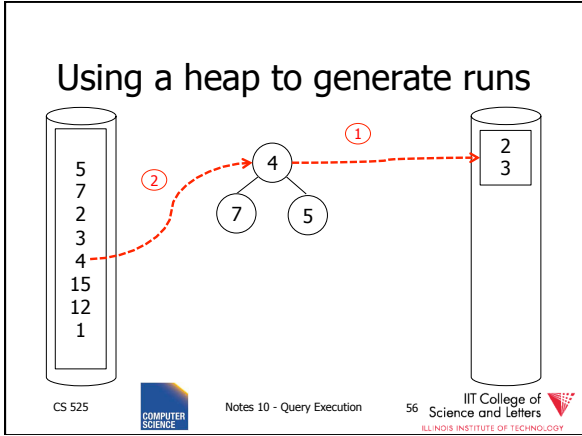
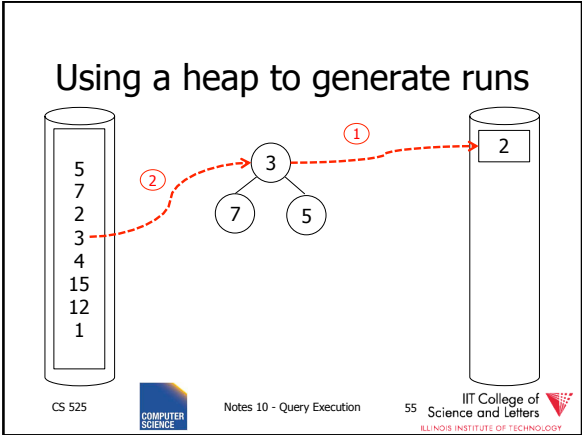
CS 525



Notes 10 - Query Execution

54

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY



Using a heap to generate runs

- Increases the run-length
 - On average by a factor of 2 (see Knuth)

CS 525



Notes 10 - Query Execution

61

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Use clustered B+-tree

- Keys in the B+-tree **I** are in sort order
 - If B+-tree is clustered traversing the leaf nodes is sequential I/O!
 - \mathbf{K} = #keys/leaf node
- Approach
 - Traverse from root to first leaf: **HT(I)**
 - Follow sibling pointers: **|R| / K**
 - Read data blocks: **B(R)**

CS 525



Notes 10 - Query Execution

62

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

I/O Operations

- HT(I) + |R| / K + B(R)** I/Os
- Less than **2 B(R)** = 1 pass external mergesort
- > Better than external merge-sort!

CS 525



Notes 10 - Query Execution

63

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Unclustered B+-tree?

- Each entry in a leaf node may point to different page of relation R
 - For each leaf page we may read up to **K** pages from relation R
 - Random I/O
- In worst-case we have
 - $\mathbf{K * B(R)}$
 - $\mathbf{K = 500}$
 - $\mathbf{500 * B(R)}$ = 250 merge passes

CS 525



Notes 10 - Query Execution

64

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Sorting Comparison

B(R) = number of block of R
M = number of available memory blocks
#RB = records per page
HT = height of B+-tree (logarithmic)
K = number of keys per leaf node

Property	Ext. Mergesort	B+ (clustered)	B+ (unclustered)
Runtime	$O(N \log_{M-1}(N))$	$O(N)$	$O(N)$
#I/O (random)	$2 B(R) * (1 + \lceil \log_{M-1}(B(R)/M) \rceil)$	$HT + R / K + B(R)$	$HT + R / K + K * \#RB$
Memory	M	1 (better HT + X)	1 (better HT + X)
Disk Space	2 B(R)	0	0
Variants	1) Merge with heap 2) Run generation with heap 3) Larger Buffer		

CS 525



Notes 10 - Query Execution

65

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Operators Overview

- (External) Sorting
- Joins (Nested Loop, Merge, Hash, ...)
- Aggregation (Sorting, Hash)
- Selection, Projection (Index, Scan)**
- Union, Set Difference
- Intersection
- Duplicate Elimination

CS 525



Notes 10 - Query Execution

66

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Scan

- Implements access to a table
 - Combined with selection
 - Probably projection too
- Variants
 - **Sequential**
 - Scan through all tuples of relation
 - **Index**
 - Use index to find tuples that match selection

CS 525



Notes 10 - Query Execution

67

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Operators Overview

- (External) Sorting
- Joins (Nested Loop, Merge, Hash, ...)
- Aggregation (Sorting, Hash)
- Selection, Projection (Index, Scan)
- Union, Set Difference
- Intersection
- Duplicate Elimination

CS 525



Notes 10 - Query Execution

68

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Options

- Transformations: $R_1 \bowtie_c R_2, R_2 \bowtie_c R_1$
- Joint algorithms:
 - Nested loop
 - Merge join
 - Join with index
 - Hash join
- Outer join algorithms

CS 525



Notes 10 - Query Execution

69

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Nested Loop Join (conceptually)

```
for each r ∈ R1 do
  for each s ∈ R2 do
    if (r,s) ⊢ C then output (r,s)
```

Applicable to:

- Any join condition C
- Cross-product

CS 525



Notes 10 - Query Execution

70

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Merge Join (conceptually)

- (1) if R_1 and R_2 not sorted, sort them
- (2) $i \leftarrow 1; j \leftarrow 1;$
While $(i \leq T(R_1)) \wedge (j \leq T(R_2))$ do
 if $R_1\{i\}.C = R_2\{j\}.C$ then outputTuples
 else if $R_1\{i\}.C > R_2\{j\}.C$ then $j \leftarrow j+1$
 else if $R_1\{i\}.C < R_2\{j\}.C$ then $i \leftarrow i+1$

Applicable to:

- C is conjunction of equalities or $</>$:
 $A_1 = B_1 \text{ AND } \dots \text{ AND } A_n = B_n$

CS 525



Notes 10 - Query Execution

71

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Procedure Output-Tuples

```
While  $(R_1\{i\}.C = R_2\{j\}.C) \wedge (i \leq T(R_1))$  do
  [jj ← j;
  while  $(R_1\{i\}.C = R_2\{jj\}.C) \wedge (jj \leq T(R_2))$  do
    [output pair  $R_1\{i\}, R_2\{jj\}$ ;
    jj ← jj+1 ]
  i ← i+1 ]
```

CS 525



Notes 10 - Query Execution

72

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Example

i	$R_1\{i\}.C$	$R_2\{j\}.C$	j
1	10	5	1
2	20	20	2
3	20	20	3
4	30	30	4
5	40	30	5
		50	6
		52	7

CS 525



Notes 10 - Query Execution

73

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Index nested loop (Conceptually)

```

For each r ∈ R1 do
    [ X ← index (R2, C, r.C)
    for each s ∈ X do
        output (r,s) pair]
    
```

Note: $X \leftarrow \text{index}(\text{rel}, \text{attr}, \text{value})$
then $X = \text{set of rel tuples with attr} = \text{value}$

CS 525



Notes 10 - Query Execution

74

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Hash join (conceptual)

Hash function h , range $0 \rightarrow k$
Buckets for R_1 : G_0, G_1, \dots, G_k
Buckets for R_2 : H_0, H_1, \dots, H_k

Applicable to:

- C is conjunction of equalities
 $A_1 = B_1 \text{ AND } \dots \text{ AND } A_n = B_n$

CS 525



Notes 10 - Query Execution

75

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Hash join (conceptual)

Hash function h , range $0 \rightarrow k$
Buckets for R_1 : G_0, G_1, \dots, G_k
Buckets for R_2 : H_0, H_1, \dots, H_k

Algorithm

- (1) Hash R_1 tuples into G buckets
- (2) Hash R_2 tuples into H buckets
- (3) For $i = 0$ to k do
match tuples in G_i, H_i buckets

CS 525



Notes 10 - Query Execution

76

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Simple example hash: even/odd

R_1	R_2	Buckets	
2	5	Even: 2 4 8	4 12 8 14
4	4		R_1 R_2
3	12	Odd: 3 5 9	5 3 13 11
5	3		
8	13		
9	8		
	11		
	14		

CS 525



Notes 10 - Query Execution

77

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Factors that affect performance

- (1) Tuples of relation stored physically together?
- (2) Relations sorted by join attribute?
- (3) Indexes exist?

CS 525



Notes 10 - Query Execution

78

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Example 1(a) NL Join $R_1 \bowtie R_2$

- Relations not contiguous
- Recall $\begin{cases} T(R_1) = 10,000 & T(R_2) = 5,000 \\ S(R_1) = S(R_2) = 1/10 \text{ block} \\ \text{MEM} = 101 \text{ blocks} \end{cases}$

CS 525



Notes 10 - Query Execution

79

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Example 1(a)

Nested Loop Join $R_1 \bowtie R_2$

- Relations not contiguous
- Recall $\begin{cases} T(R_1) = 10,000 & T(R_2) = 5,000 \\ S(R_1) = S(R_2) = 1/10 \text{ block} \\ \text{MEM} = 101 \text{ blocks} \end{cases}$

Cost: for each R_1 tuple:

[Read tuple + Read R_2]

Total = $10,000 [1 + 500] = 5,010,000$ IOs

CS 525



Notes 10 - Query Execution

80

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

- Can we do better?

CS 525



Notes 10 - Query Execution

81

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

- Can we do better?

Use our memory

- (1) Read 100 blocks of R_1
- (2) Read all of R_2 (using 1 block) + join
- (3) Repeat until done

CS 525



Notes 10 - Query Execution

82

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Cost: for each R_1 chunk:

Read chunk: 100 IOs
Read R_2 : $\frac{500 \text{ IOs}}{600}$

CS 525



Notes 10 - Query Execution

83

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Cost: for each R_1 chunk:

Read chunk: 100 IOs
Read R_2 : $\frac{500 \text{ IOs}}{600}$

Total = $\frac{1,000}{100} \times 600 = 6,000$ IOs

CS 525



Notes 10 - Query Execution

84

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

- Can we do better?

CS 525



Notes 10 - Query Execution

85

- Can we do better?

• Reverse join order: $R_2 \bowtie R_1$

$$\text{Total} = \frac{500}{100} \times (100 + 1,000) =$$

$$5 \times 1,100 = 5,500 \text{ IOs}$$

CS 525



Notes 10 - Query Execution

86

Cost of Block Nested Loop

• Reverse join order: $R_1 \bowtie R_2$

$$\text{Total} = \frac{B(R_1)}{M-1} \times (\min(B(R_1), M-1) + B(R_2))$$

CS 525



Notes 10 - Query Execution

87

Block-Nested Loop Join (conceptual)

for each M-1 blocks of R_1 do
 read M-1 blocks of R_1 into buffer
 for each block of R_2 do
 read next block of R_2
 for each tuple r in R_1 block
 for each tuple s in R_2 block
 if $(r,s) \models C$ then output (r,s)

CS 525



Notes 10 - Query Execution

88

Note

- How much memory for buffering inner and for outer chunks?
 - 1 for inner would minimize I/O
 - But, larger buffer better for I/O

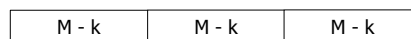
CS 525



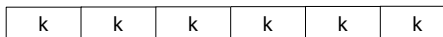
Notes 10 - Query Execution

89

R_1



R_2



CS 525



Notes 10 - Query Execution

90

Example 1(b) Merge Join

- Both R_1, R_2 ordered by C ; relations contiguous

Memory



CS 525



Notes 10 - Query Execution

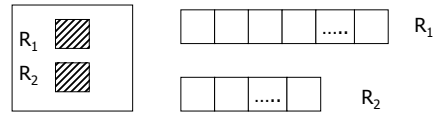
91

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Example 1(b) Merge Join

- Both R_1, R_2 ordered by C ; relations contiguous

Memory



Total cost: Read R_1 cost + read R_2 cost
= 1000 + 500 = 1,500 IOs

CS 525



Notes 10 - Query Execution

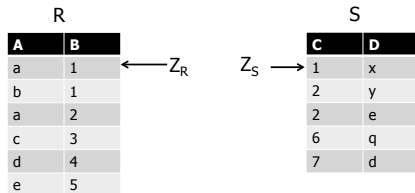
92

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Merge Join Example

$R \bowtie_{B=C} S$

Output: (a,1,1,X)



CS 525



Notes 10 - Query Execution

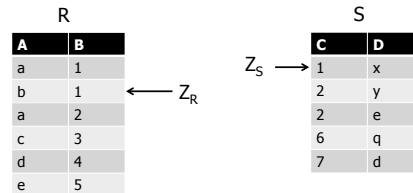
93

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Merge Join Example

$R \bowtie_{B=C} S$

Output: (b,1,1,X)



CS 525



Notes 10 - Query Execution

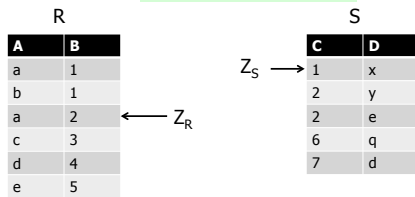
94

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Merge Join Example

$R \bowtie_{B=C} S$

R.B > S.C: advance Z_S



CS 525



Notes 10 - Query Execution

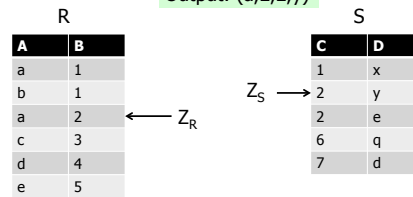
95

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Merge Join Example

$R \bowtie_{B=C} S$

Output: (a,2,2,y)



CS 525



Notes 10 - Query Execution

96

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY


Merge Join Example

$R \bowtie_{B=C} S$

Output: (a,2,2,e)

R		S	
A	B	C	D
a	1	1	x
b	1	2	y
a	2	2	e
c	3	6	q
d	4	7	d
e	5		

Diagram showing pointers Z_R and Z_S pointing to the row (a, 2) in R and (2, e) in S respectively.

CS 525  Notes 10 - Query Execution 97 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY


Merge Join Example

$R \bowtie_{B=C} S$

R.B > S.C: advance Z_S

R		S	
A	B	C	D
a	1	1	x
b	1	2	y
a	2	2	e
c	3	6	q
d	4	7	d
e	5		

Diagram showing pointer Z_S advancing to the next row (2, e) in S, while Z_R remains at (a, 2) in R.

CS 525  Notes 10 - Query Execution 98 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY


Merge Join Example

$R \bowtie_{B=C} S$

R.B < S.C: advance Z_R

R		S	
A	B	C	D
a	1	1	x
b	1	2	y
a	2	2	e
c	3	6	q
d	4	7	d
e	5		

Diagram showing pointer Z_R advancing to the next row (a, 2) in R, while Z_S remains at (2, e) in S.

CS 525  Notes 10 - Query Execution 99 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY


Merge Join Example

$R \bowtie_{B=C} S$

R.B < S.C: advance Z_R

R		S	
A	B	C	D
a	1	1	x
b	1	2	y
a	2	2	e
c	3	6	q
d	4	7	d
e	5		

Diagram showing pointer Z_R advancing to the next row (c, 3) in R, while Z_S remains at (2, e) in S.

CS 525  Notes 10 - Query Execution 100 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY


Merge Join Example

$R \bowtie_{B=C} S$

R.B < S.C: **DONE**

R		S	
A	B	C	D
a	1	1	x
b	1	2	y
a	2	2	e
c	3	6	q
d	4	7	d
e	5		


Diagram showing pointer Z_R at the end of R (row e, B=5) and Z_S at the end of S (row 7, D=d).

CS 525  Notes 10 - Query Execution 101 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

Example 1(c) Merge Join

- R_1, R_2 not ordered, but contiguous

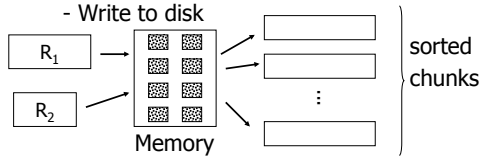
--> Need to sort R_1, R_2 first

CS 525  Notes 10 - Query Execution 102 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

One way to sort: Merge Sort

(i) For each 100 blk chunk of R:

- Read chunk
- Sort in memory
- Write to disk



CS 525

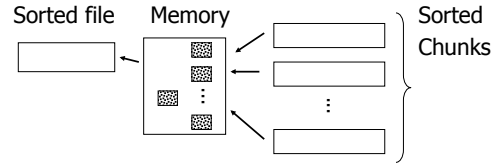


Notes 10 - Query Execution

103

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

(ii) Read all chunks + merge + write out



CS 525



Notes 10 - Query Execution

104

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Cost: Sort

Each tuple is read, written,
read, written

so...

$$\text{Sort cost } R_1: 4 \times 1,000 = 4,000$$

$$\text{Sort cost } R_2: 4 \times 500 = 2,000$$

CS 525



Notes 10 - Query Execution

105

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Example 1(d) Merge Join (continued)

R_1, R_2 contiguous, but unordered

$$\begin{aligned} \text{Total cost} &= \text{sort cost} + \text{join cost} \\ &= 6,000 + 1,500 = 7,500 \text{ IOs} \end{aligned}$$

CS 525



Notes 10 - Query Execution

106

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Example 1(c) Merge Join (continued)

R_1, R_2 contiguous, but unordered

$$\begin{aligned} \text{Total cost} &= \text{sort cost} + \text{join cost} \\ &= 6,000 + 1,500 = 7,500 \text{ IOs} \end{aligned}$$

But: Iteration cost = 5,500
so merge join does not pay off!

CS 525



Notes 10 - Query Execution

107

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

But say $R_1 = 10,000$ blocks contiguous
 $R_2 = 5,000$ blocks not ordered

$$\begin{aligned} \text{Iterate: } \frac{5000}{100} \times (100 + 10,000) &= 50 \times 10,100 \\ &= 505,000 \text{ IOs} \end{aligned}$$

$$\text{Merge join: } 5(10,000 + 5,000) = 75,000 \text{ IOs}$$

Merge Join (with sort) WINS!

CS 525



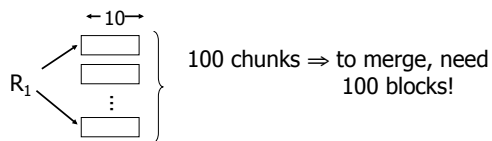
Notes 10 - Query Execution

108

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

How much memory do we need for merge sort?

E.g: Say I have 10 memory blocks



CS 525



Notes 10 - Query Execution

109

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

In general:

Say k blocks in memory

x blocks for relation sort

chunks = (x/k) size of chunk = k

CS 525



Notes 10 - Query Execution

110

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

In general:

Say k blocks in memory

x blocks for relation sort

chunks = (x/k) size of chunk = k

chunks < buffers available for merge

CS 525



Notes 10 - Query Execution

111

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

In general:

Say k blocks in memory

x blocks for relation sort

chunks = (x/k) size of chunk = k

chunks < buffers available for merge

so... $(x/k) \leq k$

or $k^2 \geq x$ or $k \geq \sqrt{x}$

CS 525



Notes 10 - Query Execution

112

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

In our example

R_1 is 1000 blocks, $k \geq 31.62$

R_2 is 500 blocks, $k \geq 22.36$

Need at least 32 buffers

Again: in practice we would not want to use only one buffer per run!

CS 525



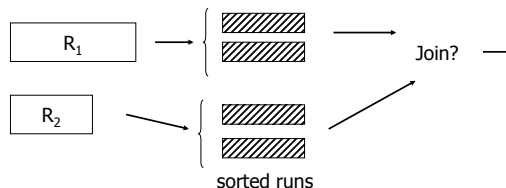
Notes 10 - Query Execution

113

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Can we improve on merge join?

Hint: do we really need the fully sorted files?



CS 525



Notes 10 - Query Execution

114

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Cost of improved merge join:

$C = \text{Read } R_1 + \text{write } R_1 \text{ into runs}$
 $+ \text{read } R_2 + \text{write } R_2 \text{ into runs}$
 $+ \text{join}$
 $= 2,000 + 1,000 + 1,500 = 4,500$

--> Memory requirement?

CS 525



Notes 10 - Query Execution

115

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Example 1(d) Index Join

- Assume $R_1.C$ index exists; 2 levels
- Assume R_2 contiguous, unordered
- Assume $R_1.C$ index fits in memory

CS 525



Notes 10 - Query Execution

116

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Cost: Reads: 500 IOs
for each R_2 tuple:

- probe index - free
- if match, read R_1 tuple: 1 IO

CS 525



Notes 10 - Query Execution

117

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

What is expected # of matching tuples?

(a) say $R_1.C$ is key, $R_2.C$ is foreign key
then expect = 1

(b) say $V(R_1.C) = 5000$, $T(R_1) = 10,000$
with uniform assumption
expect = $10,000/5,000 = 2$

CS 525



Notes 10 - Query Execution

118

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

What is expected # of matching tuples?

(c) Say $\text{DOM}(R_1, C) = 1,000,000$
 $T(R_1) = 10,000$
with alternate assumption
Expect = $\frac{10,000}{1,000,000} = \frac{1}{100}$

CS 525



Notes 10 - Query Execution

119

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Total cost with index join

(a) Total cost = $500 + 5000(1)1 = 5,500$

(b) Total cost = $500 + 5000(2)1 = 10,500$

(c) Total cost = $500 + 5000(1/100)1 = 550$

CS 525



Notes 10 - Query Execution

120

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

What if index does not fit in memory?

Example: say $R_1.C$ index is 201 blocks

- Keep root + 99 leaf nodes in memory
- Expected cost of each probe is

$$E = (0)\frac{99}{200} + (1)\frac{101}{200} \approx 0.5$$

CS 525



Notes 10 - Query Execution

121

Total cost (including probes)

$$\begin{aligned} &= 500+5000 \text{ [Probe + get records]} \\ &= 500+5000 [0.5+2] \quad \text{uniform assumption} \\ &= 500+12,500 = 13,000 \quad \text{(case b)} \end{aligned}$$

CS 525



Notes 10 - Query Execution

122

Total cost (including probes)

$$\begin{aligned} &= 500+5000 \text{ [Probe + get records]} \\ &= 500+5000 [0.5+2] \quad \text{uniform assumption} \\ &= 500+12,500 = 13,000 \quad \text{(case b)} \end{aligned}$$

For case (c):

$$\begin{aligned} &= 500+5000[0.5 \times 1 + (1/100) \times 1] \\ &= 500+2500+50 = 3050 \text{ IOs} \end{aligned}$$

CS 525



Notes 10 - Query Execution

123

So far

{	Nested Loop	5500	
	Merge join	1500	
	Sort+Merge Join	7500	→ 4500
	$R_1.C$ Index	5500	→ 3050 → 550
	$R_2.C$ Index	_____	

CS 525

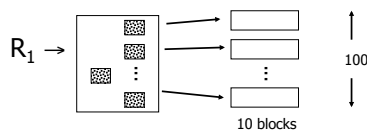


Notes 10 - Query Execution

124

Example 1(e) Partition Hash Join

- R_1, R_2 contiguous (un-ordered)
- Use 100 buckets
- Read R_1 , hash, + write buckets



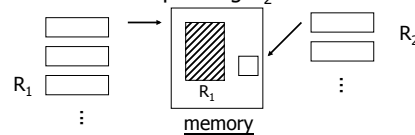
CS 525



Notes 10 - Query Execution

125

- > Same for R_2
- > Read one R_1 bucket; build memory hash table
 - using different hash function h'
- > Read corresponding R_2 bucket + hash probe



⊞ Then repeat for all buckets

CS 525



Notes 10 - Query Execution

126

Cost:

“Bucketize:” Read R_1 + write

Read R_2 + write

Join: Read R_1, R_2

Total cost = $3 \times [1000+500] = 4500$

CS 525



Notes 10 - Query Execution

127

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Cost:

“Bucketize:” Read R_1 + write

Read R_2 + write

Join: Read R_1, R_2

Total cost = $3 \times [1000+500] = 4500$

Note: this is an approximation since buckets will vary in size and we have to round up to blocks

CS 525

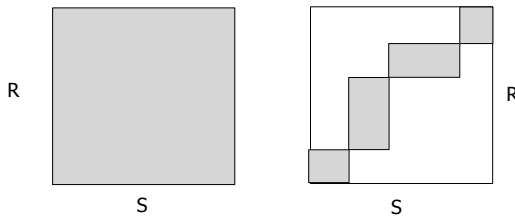


Notes 10 - Query Execution

128

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Why is Hash Join good?



CS 525



Notes 10 - Query Execution

129

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Minimum memory requirements:

Size of R_1 bucket = (x/k)

k = number of memory buffers

x = number of R_1 blocks

So... $(x/k) < k$

$k > \sqrt{x}$ need: $k+1$ total memory buffers

CS 525



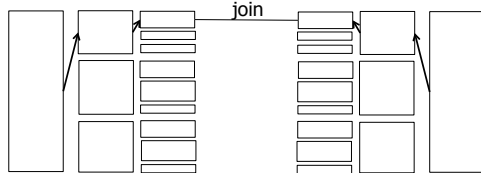
Notes 10 - Query Execution

130

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Can we use Hash-join when buckets do not fit into memory?:

- Treat buckets as relations and apply Hash-join recursively



CS 525



Notes 10 - Query Execution

131

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Duality Hashing-Sorting

- Both partition inputs
- Until input fits into memory
- Logarithmic number of phases in memory size

CS 525



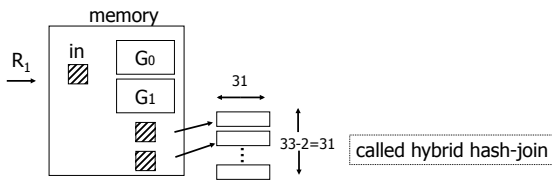
Notes 10 - Query Execution

132

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Trick: keep some buckets in memory

E.g., $k' = 33$ R_1 buckets = 31 blocks
keep 2 in memory



CS 525



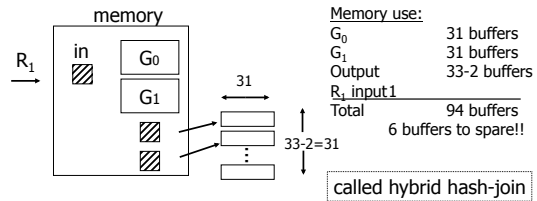
Notes 10 - Query Execution

133

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Trick: keep some buckets in memory

E.g., $k' = 33$ R_1 buckets = 31 blocks
keep 2 in memory



CS 525



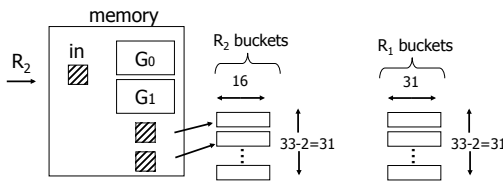
Notes 10 - Query Execution

134

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Next: Bucketize R_2

- R_2 buckets = $500/33 = 16$ blocks
- Two of the R_2 buckets joined immediately with G_0, G_1



CS 525



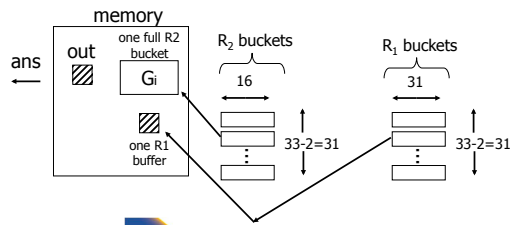
Notes 10 - Query Execution

135

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Finally: Join remaining buckets

- for each bucket pair:
 - read one of the buckets into memory
 - join with second bucket



CS 525



Notes 10 - Query Execution

136

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Cost

- Bucketize $R_1 = 1000 + 31 \times 31 = 1961$
- To bucketize R_2 , only write 31 buckets:
so, cost = $500 + 31 \times 16 = 996$
- To compare join (2 buckets already done)
read $31 \times 31 + 31 \times 16 = 1457$

Total cost = $1961 + 996 + 1457 = 4414$

CS 525

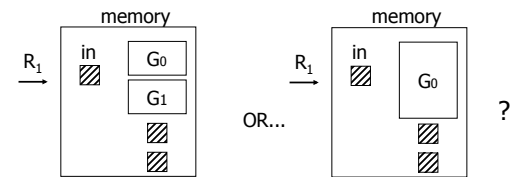


Notes 10 - Query Execution

137

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

• How many buckets in memory?



• See textbook for answer...

CS 525



Notes 10 - Query Execution

138

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Another hash join trick:

- Only write into buckets <val,ptr> pairs
- When we get a match in join phase, must fetch tuples

CS 525



Notes 10 - Query Execution

139



- To illustrate cost computation, assume:
 - 100 <val,ptr> pairs/block
 - expected number of result tuples is 100

CS 525



Notes 10 - Query Execution

140



- To illustrate cost computation, assume:
 - 100 <val,ptr> pairs/block
 - expected number of result tuples is 100
- Build hash table for R_2 in memory
5000 tuples \rightarrow $5000/100 = 50$ blocks
- Read R_1 and match
- Read $\sim 100 R_2$ tuples

CS 525



Notes 10 - Query Execution

141



- To illustrate cost computation, assume:
 - 100 <val,ptr> pairs/block
 - expected number of result tuples is 100
- Build hash table for R_2 in memory
5000 tuples \rightarrow $5000/100 = 50$ blocks
- Read R_1 and match
- Read $\sim 100 R_2$ tuples

<u>Total cost</u> =	Read R_2 :	500
	Read R_1 :	1000
	Get tuples:	<u>100</u>
		1600

CS 525



Notes 10 - Query Execution

142



So far:

Iterate	5500
Merge join	1500
Sort+merge joint	7500
$R_1.C$ index	5500 \rightarrow 550
$R_2.C$ index	_____
Build $R_1.C$ index	_____
Build $R_2.C$ index	_____
Hash join	4500+
with trick, R_1 first	4414
with trick, R_2 first	_____
Hash join, pointers	1600

CS 525



Notes 10 - Query Execution

143



Yet another hash join trick:

- Combine the ideas of
 - block nested-loop with hash join
- Use memory to build hash-table for one chunk of relation
- Find join partners in $O(1)$ instead of $O(M)$
- Trade-off
 - Space-overhead of hash-table
 - Time savings from look-up

CS 525



Notes 10 - Query Execution

144



Summary

- Nested Loop ok for “small” relations (relative to memory size)
 - Need for complex join condition
- For equi-join, where relations not sorted and no indexes exist, hash join usually best

CS 525



Notes 10 - Query Execution

145

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

- Sort + merge join good for non-equi-join (e.g., $R_1.C > R_2.C$)
- If relations already sorted, use merge join
- If index exists, it could be useful (depends on expected result size)

CS 525



Notes 10 - Query Execution

146

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Join Comparison

N_i = number of tuples in R_i
 $B(R_i)$ = number of blocks of R_i
 $\#P$ = number of partition steps for hash join
 P_{ij} = average number of join partners

Algorithm	#I/O	Memory	Disk Space
Nested Loop (block)	$B(R_1) / (M-1) * [\min(B(R_1), M-1) + B(R_2)]$	3	0
Index Nested Loop	$B(R_1) + N_1 * P_{12}$	$B(\text{Index}) + 2$	0
Merge (sorted)	$B(R_1) + B(R_2)$	Max tuples =	0
Merge (unsorted)	$B(R_1) + B(R_2) + (\text{sort} - 1 \text{ pass})$	sort	$B(R_1) + B(R_2)$
Hash	$(2\#P + 1) (B(R_1) + B(R_2))$	$\text{root}(\max(B(R_1), B(R_2)), \#P + 1)$	$\sim B(R_1) + B(R_2)$

CS 525



Notes 10 - Query Execution

147

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Why do we need nested loop?

- Remember not all join implementations work for all types of join conditions

Algorithm	Type of Condition	Example
Nested Loop	any	a LIKE '%hello%'
Index Nested Loop	Supported by index: Equi-join (hash) Equi or range (B-tree)	a = b a < b
Merge	Equalities and ranges	a < b, a = b AND c = d
Hash	Equi-join	a = b

CS 525



Notes 10 - Query Execution

148

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Outer Joins

- How to implement (left) outer joins?
- Nested Loop and Merge
 - Use a flag that is set to true if we find a match for an outer tuple
 - If flag is false fill with NULL
- Hash
 - If no matching tuple fill with NULL

CS 525

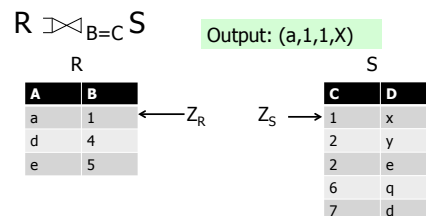


Notes 10 - Query Execution

149

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Merge Left Outer Join



CS 525



Notes 10 - Query Execution

150

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Merge Left Outer Join

$R \bowtie_{B=C} S$

R

A	B
a	1
d	4
e	5

No match for (d,4)
Output: (d,4,NULL,NULL)

S

C	D
1	x
2	y
2	e
6	q
7	d

CS 525



Notes 10 - Query Execution

151

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Merge Left Outer Join

$R \bowtie_{B=C} S$

R

A	B
a	1
d	4
e	5

No match for (e,5)
Output: (e,5,NULL,NULL)

S

C	D
1	x
2	y
2	e
6	q
7	d

CS 525



Notes 10 - Query Execution

152

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Operators Overview

- (External) Sorting
- Joins (Nested Loop, Merge, Hash, ...)
- **Aggregation (Sorting, Hash)**
- Selection, Projection (Index, Scan)
- Union, Set Difference
- Intersection
- Duplicate Elimination

CS 525



Notes 10 - Query Execution

153

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Aggregation

- Have to compute aggregation functions
 - for each group of tuples from input
- Groups
 - Determined by equality of group-by attributes

CS 525



Notes 10 - Query Execution

154

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Aggregation Example

```
SELECT sum(a), b
FROM R
GROUP BY b
```

a	b
3	1
4	2
3	1
1	2
1	2

sum(a)	b
6	1
6	2

CS 525



Notes 10 - Query Execution

155

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Aggregation Function Interface

- `init()`
 - Initialize state
- `update(tuple)`
 - Update state with information from tuple
- `close()`
 - Return result and clean-up

CS 525



Notes 10 - Query Execution

156

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Implementation SUM(A)

- `init()`
 - `sum := 0`
- `update(tuple)`
 - `sum += tuple.A`
- `close()`
 - **return** `sum`

CS 525



Notes 10 - Query Execution

157

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Aggregation Implementations

- **Sorting**
 - Sort input on group-by attributes
 - On group boundaries output tuple
- **Hashing**
 - Store current aggregated values for each group in hash table
 - Update with newly arriving tuples
 - Output result after processing all inputs

CS 525



Notes 10 - Query Execution

158

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Grouping by sorting

- Similar to Merge join
- Sort R on group-by attribute
- Scan through sorted input
 - If group-by values change
 - Output using `close()` and call `init()`
 - Otherwise
 - Call `update()`

CS 525



Notes 10 - Query Execution

159

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Aggregation Example

```
SELECT sum(a), b
FROM R
GROUP BY b
```

sort

a	b	a	b
3	1	3	1
4	2	3	1
3	1	4	2
1	2	1	2
1	2	1	2

`init()`
0

CS 525



Notes 10 - Query Execution

160

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Aggregation Example

```
SELECT sum(a), b
FROM R
GROUP BY b
```

a	b
3	1
3	1
4	2
1	2
1	2

`update(3,1)`
3

CS 525



Notes 10 - Query Execution

161

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Aggregation Example

```
SELECT sum(a), b
FROM R
GROUP BY b
```

a	b
3	1
3	1
4	2
1	2
1	2

`update(3,1)`
6

CS 525



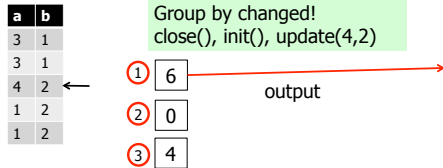
Notes 10 - Query Execution

162

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Aggregation Example

```
SELECT sum(a), b
FROM R
GROUP BY b
```



CS 525



Notes 10 - Query Execution

163

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Grouping by Hashing

- Create in-memory hash-table
- For each input tuple probe hash table with group by values
 - If no entry exists then call init(), update(), and add entry
 - Otherwise call update() for entry
- Loop through all entries in hash-table and output calling close()

CS 525



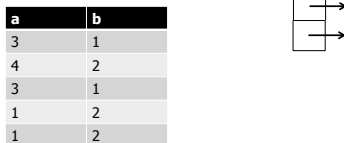
Notes 10 - Query Execution

164

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Aggregation Example

```
SELECT sum(a), b
FROM R
GROUP BY b
```



CS 525



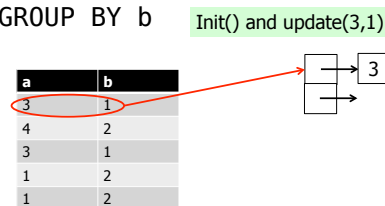
Notes 10 - Query Execution

165

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Aggregation Example

```
SELECT sum(a), b
FROM R
GROUP BY b
```



CS 525



Notes 10 - Query Execution

166

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Aggregation Example

```
SELECT sum(a), b
FROM R
GROUP BY b
```



CS 525



Notes 10 - Query Execution

167

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Aggregation Example

```
SELECT sum(a), b
FROM R
GROUP BY b
```



CS 525



Notes 10 - Query Execution

168

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Aggregation Example

```
SELECT sum(a), b
FROM R
GROUP BY b
```

- Loop through hash table entries
- Output tuples

a	b
3	1
4	2
3	1
1	2
1	2



CS 525



Notes 10 - Query Execution

169

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Aggregation Summary

- Hashing
 - No sorting -> no extra I/O
 - Hash table has to fit into memory
 - No outputs before all inputs have been processed
- Sorting
 - No memory required
 - Output one group at a time

CS 525



Notes 10 - Query Execution

170

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Operators Overview

- (External) Sorting
- Joins (Nested Loop, Merge, Hash, ...)
- Aggregation (Sorting, Hash)
- Selection, Projection (Index, Scan)
- Union, Set Difference
- Intersection
- Duplicate Elimination

CS 525



Notes 10 - Query Execution

171

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Duplicate Elimination

- Equivalent to group-by on all attributes
- -> Can use aggregation implementations
- Optimization
 - Hash
 - Directly output tuple and use hash table only to avoid outputting duplicates

CS 525



Notes 10 - Query Execution

172

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Operators Overview

- (External) Sorting
- Joins (Nested Loop, Merge, Hash, ...)
- Aggregation (Sorting, Hash)
- Selection, Projection (Index, Scan)
- Union, Set Difference
- Intersection
- Duplicate Elimination

CS 525



Notes 10 - Query Execution

173

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Set Operations

- Can be modeled as join
 - with different output requirements
- As aggregation/group by on all columns
 - with different output requirements

CS 525



Notes 10 - Query Execution

174

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Union

- Bag union
 - Append the two inputs
 - E.g., using three buffers
- Set union
 - Apply duplicate removal to result

CS 525



Notes 10 - Query Execution

175

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Intersection

- Set version
 - Equivalent to join + project + duplicate removal
 - 3-state aggregate function (found left, found right, found both)
- Bag version
 - Join + project + $\min(i,j)$
 - Aggregate $\min(\text{count}(i), \text{count}(j))$

CS 525



Notes 10 - Query Execution

176

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Set Difference

- Using join methods
 - Find matching tuples
 - If no match found, then output
- Using aggregation
 - $\text{count}(i) - \text{count}(j)$ (**bag**)
 - $\text{true}(i) \text{ AND } \text{false}(j)$ (**set**)

CS 525



Notes 10 - Query Execution

177

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Summary

- Operator implementations
 - Joins!
 - Other operators
- Cost estimations
 - I/O
 - memory
- Query processing architectures

CS 525



Notes 10 - Query Execution

178

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Next

- Query Optimization Physical
- -> How to **efficiently** choose an **efficient** plan

CS 525



Notes 10 - Query Execution

179

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY



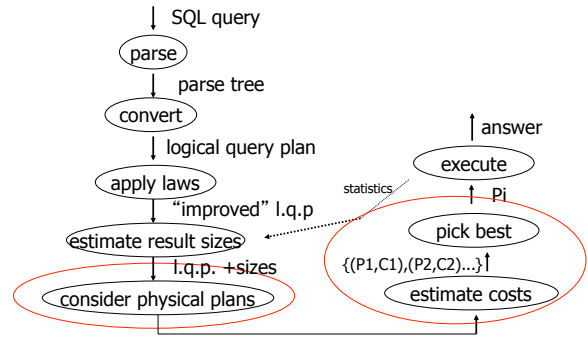
CS 525: Advanced Database Organization

11: Query Optimization

Physical

Boris Glavic

Slides: adapted from a [course](#) taught by [Hector Garcia-Molina](#), Stanford InfoLab



Cost of Query

- Parse + Analyze
- Optimization – Find plan
- Execution
- Return results to client

Cost of Query

- Parse + Analyze
 - Can parse MB of SQL code in miliseconds
- **Optimization – Find plan**
 - **Generating plans, costing plans**
- **Execution**
 - **Execute plan**
- Return results to client
 - Can be expensive but not discussed here

Physical Optimization

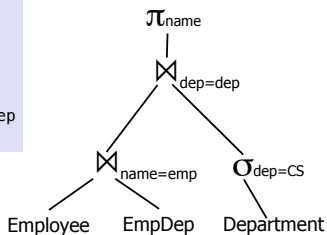
- Apply after applying heuristics in logical optimization
- 1) Enumerate potential execution plans
 - All?
 - Subset
- 2) Cost plans
 - What cost function?

Physical Optimization

- To apply pruning in the search for the best plan
 - Steps 1 and 2 have to be interleaved
 - Prune parts of the search space
 - if we know that it cannot contain any plan that is better than what we found so far

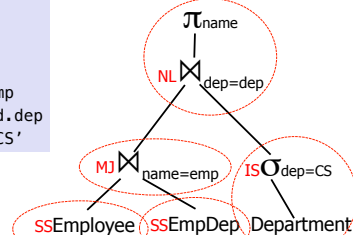
Example Query

```
SELECT e.name
FROM Employee e,
      EmpDep ed,
      Department d
WHERE e.name = ed.emp
      AND ed.dep = d.dep
      AND d.dep = 'CS'
```



Example Query – Possible Plan

```
SELECT e.name
FROM Employee e,
      EmpDep ed,
      Department d
WHERE e.name = ed.emp
      AND ed.dep = d.dep
      AND d.dep = 'CS'
```

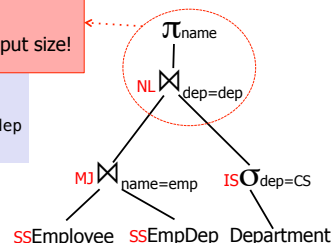


Cost Model

- Cost factors
 - #disk I/O
 - CPU cost
 - Response time
 - Total **execution time**
- Cost of operators
 - I/O as discussed in query execution (part 10)
 - Need to know **size of intermediate results** (part 09)

Example Query – Possible Plan

```
SELECT e.name
FROM Employee e,
      EmpDep ed,
      Department d
WHERE e.name = ed.emp
      AND ed.dep = d.dep
      AND d.dep = 'CS'
```



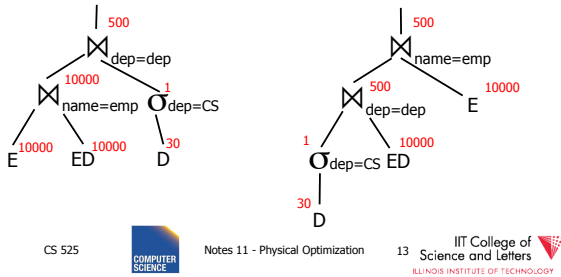
Cost Model Trade-off

- **Precision**
 - Incorrect cost-estimation -> choose suboptimal plan
- **Cost of computing cost**
 - Cost of costing a plan
 - We may have to cost millions or billions of plans
 - Cost of maintaining statistics
 - Occupies resources needed for query processing

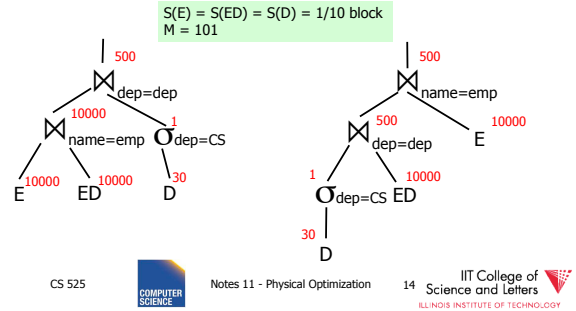
Plan Enumeration

- For each operator in the query
 - Several implementation options
- Binary operators (joins)
 - Changing the order may improve performance a lot!
- -> consider both **different implementations and order of operators** in plan enumeration

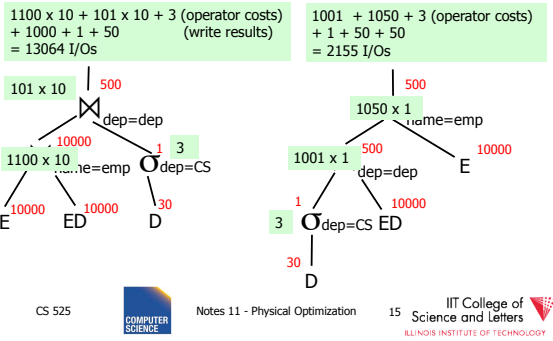
Example Join Ordering Result Sizes



Example Join Ordering Cost (only NL)



$S(E) = S(ED) = S(D) = 1/10$ block
 $M = 101$
I/O costs only
No pipelining, write all results to disk



Plan Enumeration

- All
 - Consider all potential plans of a certain type (discussed later)
 - Prune only if sure
- Heuristics
 - Apply heuristics to prune search space
- Randomized Algorithms

Plan Enumeration Algorithms

- All
 - Dynamic Programming (System R)
 - A* search
- Heuristics
 - Minimum Selectivity, Intermediate result size, ...
 - KBZ-Algorithm, AB-Algorithm
- Randomized
 - Genetic Algorithms
 - Simulated Annealing

Reordering Joins Revisited

- Equivalences (Natural Join)
 1. $R \bowtie S \equiv S \bowtie R$
 2. $(R \bowtie S) \bowtie T \equiv R \bowtie (S \bowtie T)$
- Equivalences Equi-Join
 1. $R \bowtie_{a=b} S \equiv S \bowtie_{a=b} R$
 2. $(R \bowtie_{a=b} S) \bowtie_{c=d} T \equiv R \bowtie_{a=b} (S \bowtie_{c=d} T)$
 3. $\sigma_{a=b}(R \times S) \equiv R \bowtie_{a=b} S$

Equi-Join Equivalences

- $(R \bowtie_{a=b} S) \bowtie_{c=d} T \equiv R \bowtie_{a=b} (S \bowtie_{c=d} T)$
 - What if c is attribute of R ?
- $(R \bowtie_{a=b} S) \bowtie_{c=d} T \equiv R \bowtie_{a=b \wedge c=d} (S \times T)$
- $\sigma_{a=b} (R \times S) \equiv R \bowtie_{a=b} S$
 - Only useful if a is from R and S from b (vice-versa)

Why Cross-Products are bad

- We discussed efficient join algorithms
 - Merge-join $O(n)$ resp. $O(n \log(n))$
 - Vs. Nested-loop $O(n^2)$
- $R \times S$
 - Result size is $O(n^2)$
 - Cannot be better than $O(n^2)$
 - Surprise, surprise: merge-join doesn't work
no need to sort, but degrades to nested loop

Agenda

- Given some query
 - How to enumerate all plans?
- Try to avoid cross-products
- Need way to figure out if equivalences can be applied
 - Data structure: **Join Graph**

Join Graph

- Assumptions
 - Only equi-joins ($a = b$)
 - a and b are either constants or attributes
 - Only conjunctive join conditions (AND)

Join Graph

- Nodes: Relations R_1, \dots, R_n of query
- Edges: Join conditions
 - Add edge between R_i and R_j labeled with C
 - if there is a join condition C
 - That equates an attribute from R_i with an attribute from R_j
 - Add a self-edge to R_i for each simple predicate

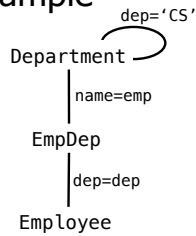
Join Graph Example

<pre>SELECT e.name FROM Employee e, EmpDep ed, Department d WHERE e.name = ed.emp AND ed.dep = d.dep AND d.dep = 'CS'</pre>	<p>Department</p> <p>EmpDep</p> <p>Employee</p>
---	---

Join Graph Example

```

SELECT e.name
FROM Employee e,
      EmpDep ed,
      Department d
WHERE e.name = ed.emp
      AND ed.dep = d.dep
      AND d.dep = 'CS'
    
```



CS 525



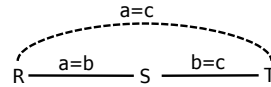
Notes 11 - Physical Optimization

25



Notes on Join Graph

- Join Graph tells us in which ways we can join without using cross products
- However, ...
 - Only if transitivity is considered



CS 525

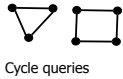
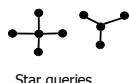


Notes 11 - Physical Optimization

26



Join Graph Shapes



CS 525



Notes 11 - Physical Optimization

27



Join Graph Shapes



```

SELECT *
FROM R, S, T
WHERE R.a = S.b
      AND S.c = T.d
    
```

CS 525



Notes 11 - Physical Optimization

28



Join Graph Shapes



Star queries

```

SELECT *
FROM R, S, T, U
WHERE R.a = S.a
      AND R.b = T.b
      AND R.c = U.c
    
```

CS 525



Notes 11 - Physical Optimization

29



Join Graph Shapes



Tree queries

```

SELECT *
FROM R, S, T, U, V
WHERE R.a = S.a
      AND R.b = T.b
      AND T.c = U.c
      AND T.d = V.d
    
```

CS 525

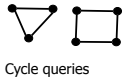


Notes 11 - Physical Optimization

30



Join Graph Shapes



Cycle queries

```
SELECT *
FROM R,S,T
WHERE R.a = S.a
AND S.b = T.b
AND T.c = R.c
```

Join Graph Shapes

```
SELECT *
FROM R,S,T
WHERE R.a = S.a
AND S.b = T.b
AND T.c = R.c
```



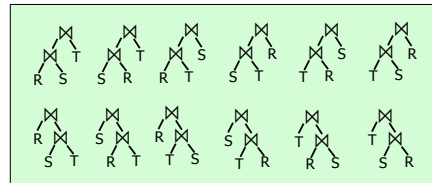
Clique queries

How many join orders?

- Assumption
 - Use cross products (can freely reorder)
 - Joins are binary operations
 - Two inputs
 - Each input either join result or relation access

How many join orders?

- Example 3 relations R,S,T
 - 12 orders



How many join orders?

- A join over $n+1$ relations requires n binary joins
- The root of the join tree joins k with $n - k - 1$ join operators ($0 \leq k \leq n-1$)



How many join orders?

- This are the **Catalan numbers**

$$C_n = \sum_{k=0}^{n-1} C_k \times C_{n-k-1} = (2n)! / (n+1)!n!$$

$$C_0 = 1$$

How many join orders?

- This are the **Catalan numbers**
- For each such tree we can permute the input relations **(n+1)!** Permutations

$$(2n)! / (n+1)!n! * (n+1)! = (2n)!/n!$$

How many join orders?

#relations	#join trees
2	2
3	12
4	120
5	1,680
6	30,240
7	665,280
8	17,297,280
9	17,643,225,600
10	670,442,572,800
11	28,158,588,057,600

How many join orders?

- If for each join we consider **k** join algorithms then for **n** relations we have
 - Multiply with a factor **kⁿ⁻¹**
- Example consider
 - Nested loop
 - Merge
 - Hash

How many join orders?

#relations	#join trees
2	6
3	108
4	3240
5	136,080
6	7,348,320
7	484,989,120
8	37,829,151,360
9	115,757,203,161,600
10	13,196,321,160,422,400
11	1,662,736,466,213,222,400

Too many join orders?

- Even if costing is cheap
 - Unrealistic assumption 1 CPU cycle
 - Realistic are thousands or millions of instructions
- Cost all join options for 11 relations
 - 3GHz CPU, 8 cores
 - 69,280,686 sec > 2 years

How to deal with excessive number of combinations?

- Prune parts based on optimality
 - Dynamic programming
 - A*-search
- Only consider certain types of join trees
 - Left-deep, Right-deep, zig-zag, bushy
- Heuristic and random algorithms

Dynamic Programming

- Assumption: **Principle of Optimality**
 - To compute the **global** optimal plan it is only necessary to consider the optimal solutions for its **sub-queries**
- Does this assumption hold?
 - Depends on cost-function

What is dynamic programming?

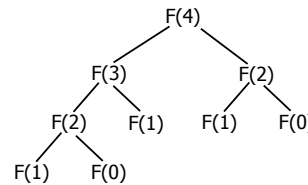
- Recall data structures and algorithms 101!
- Consider a **Divide-and-Conquer** problem
 - Solutions for a problem of size **n** can be build from solutions for sub-problems of smaller size (e.g., **n/2** or **n-1**)
- **Memoize**
 - Store solutions for sub-problems
 - -> Each solution has to be only computed once
 - -> Needs extra memory

Example Fibonacci Numbers

- $F(n) = F(n-1) + F(n-2)$
- $F(0) = F(1) = 1$

```
Fib(n)
{
    if (n = 0) return 0
    else if (n = 1) return 1
    else return Fib(n-1) + Fib(n-2)
}
```

Example Fibonacci Numbers



Complexity

- Number of calls
 - $C(n) = C(n-1) + C(n-2) + 1 = \text{Fib}(n+2)$
 - $O(2^n)$

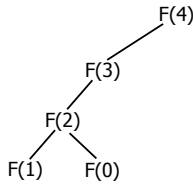
Using dynamic programming

```
Fib(n)
{
    int[] fib;
    fib[0] = 1;
    fib[1] = 1;

    for(i = 2; i < n; i++)
        fib[i] = fib[i-1] + fib[i-2]

    return fib[n];
}
```

Example Fibonacci Numbers



CS 525



Notes 11 - Physical Optimization

49



What do we gain?

- $O(n)$ instead of $O(2^n)$

CS 525



Notes 11 - Physical Optimization

50



Dynamic Programming for Join Enumeration

- Find cheapest plan for n -relation join in n passes
- For each i in $1 \dots n$
 - Construct solutions of size i from best solutions of size $< i$

CS 525



Notes 11 - Physical Optimization

51



DP Join Enumeration

```

optPlan ← Map({R}, {plan})

find_join_dp(q(R1, ..., Rn))
{
  for i=1 to n
    optPlan[{Ri}] ← access_paths(Ri)
  for i=2 to n
    foreach S ⊆ {R1, ..., Rn} with |S|=i
      optPlan[S] ← ∅
      foreach 0 ⊂ S with 0 ≠ ∅
        optPlan[S] ← optPlan[S] ∪ possible_joins(optPlan(0), optPlan(S\0))
      prune_plans(optPlan[S])
  return optPlan[{R1, ..., Rn}]
}
  
```

CS 525



Notes 11 - Physical Optimization

52



Dynamic Programming for Join Enumeration

- `access_paths (R)`
 - Find cheapest access path for relation R
- `possible_joins(plan, plan)`
 - Enumerate all joins (merge, NL, ...) variants between the input plans
- `prune_plans({plan})`
 - Only keep cheapest plan from input set

CS 525



Notes 11 - Physical Optimization

53



DP-JE Complexity

- Time: $O(3^n)$
- Space: $O(2^n)$
- Still too much for large number of joins (10-20)

CS 525

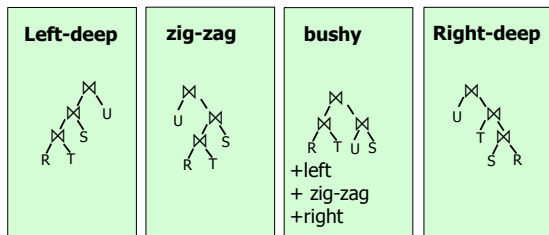


Notes 11 - Physical Optimization

54



Types of join trees



CS 525



Notes 11 - Physical Optimization

55

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Number of Join-Trees

- Number of join trees for n relations
- Left-deep: $n!$
- Right-deep: $n!$
- Zig-zag: $2^{n-2}n!$

CS 525



Notes 11 - Physical Optimization

56

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

How many join orders?

#relations	#bushy join trees	#left-deep join trees
2	2	2
3	12	6
4	120	24
5	1,680	120
6	30,240	720
7	665,280	5040
8	17,297,280	40,230
9	17,643,225,600	362,880
10	670,442,572,800	3,628,800
11	28,158,588,057,600	39,916,800

CS 525



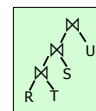
Notes 11 - Physical Optimization

57

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

DP with Left-deep trees only

- Reduced search-space
- Each join is with input relation
 - ->can use index joins
 - ->easy to pipe-line
- DP with left-deep plans was introduced by system R, the first relational database developed by IBM Research



CS 525



Notes 11 - Physical Optimization

58

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Revisiting the assumption

- Is it really sufficient to only look at the best plan for every sub-query?
- Cost of merge join depends whether the input is already sorted
 - -> A sub-optimal plan may produce results ordered in a way that reduces cost of joining above
 - Keep track of **interesting orders**

CS 525



Notes 11 - Physical Optimization

59

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Interesting Orders

- Number of interesting orders is usually small
- ->Extend DP join enumeration to keep track of interesting orders
 - Determine interesting orders
 - For each sub-query store best-plan for each interesting order

CS 525



Notes 11 - Physical Optimization

60



IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Example Interesting Orders

Left-deep best plans: 3-way {R,S,T}

Left-deep best plans: 2-way

{R,S} {R,T} {S,T}



CS 525  Notes 11 - Physical Optimization 61  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Example Interesting Orders

Left-deep best plans: 3-way {R,S,T}

Left-deep best plans: 2-way

{R,S} {R,T} {S,T}

CS 525  Notes 11 - Physical Optimization 62  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Greedy Join Enumeration

- Heuristic method
 - Not guaranteed that best plan is found
- Start from single relation plans
- In each iteration greedily join to plans with the minimal cost
- Until a plan for the whole query has been generated

Greedy Join Enumeration

```
plans ← list({plan})
find_join_dp(q(R1, ..., Rn))
{
  for i=1 to n
    plans ← plans ∪ access_paths(Ri)
  for i=n to 2
    cheapest = argminj,k∈{1,...,n} (cost(Pj ⋈ Pk))
    plans ← plans \ {Pj, Pk} ∪ {Pj ⋈ Pk}
  return plans // single plan left
}
```

Greedy Join Enumeration

- Time: $O(n^3)$
 - Loop iterations: $O(n)$
 - In each iterations looking of pairs of plans in of max size n : $O(n^2)$
- Space: $O(n^2)$
 - Needed to store the current list of plans

Randomized Join-Algorithms

- Iterative improvement
- Simulated annealing
- Tabu-search
- Genetic algorithms

Transformative Approach

- Start from (random) complete solutions
- Apply transformations to generate new solutions
 - Direct application of equivalences
 - Commutativity
 - Associativity
 - Combined equivalences
 - E.g., $(R \bowtie S) \bowtie T \equiv T \bowtie (S \bowtie R)$

Concern about Transformative Approach

- Need to be able to generate random plans fast
- Need to be able to apply transformations fast
 - Trade-off: space covered by transformations vs. number and complexity of transformation rules

Iterative Improvement

```

improve(q(R1, ..., Rn))
{
  best ← random_plan(q)
  while (not reached time limit)
    curplan ← random_plan(q)
    do
      prevplan ← curplan
      curplan ← apply_random_trans (prevplan)
      while (cost(curplan) < cost(prevplan))
        if (cost(prevplan) < cost(best))
          best ← prevplan
    return best
}
    
```

Iterative Improvement

- Easy to get stuck in local minimum
- **Idea:** Allow transformations that result in more expensive plans with the hope to move out of local minima
 - > Simulated Annealing

Simulated Annealing

```

SA(q(R1, ..., Rn))
{
  best ← random_plan(q)
  curplan ← best
  t ← tinit // "temperature"
  while (t > 0)
    newplan ← apply_random_trans(curplan)
    if cost(newplan) < cost(curplan)
      curplan ← newplan
    else if random() < e-(cost(newplan)-cost(curplan))/t
      curplan ← newplan
    if (cost(curplan) < cost(best))
      best ← curplan
    reduce(t)
  return best
}
    
```

Simulated Annealing

```

SA(q(R1, ..., Rn))
{
  best ← random_plan(q)
  curplan ← best
  t ← tinit // "temperature"
  while (t > 0)
    newplan ← apply_random_trans(curplan)
    if cost(newplan) < cost(curplan)
      curplan ← newplan
    else if random() < e-(cost(newplan)-cost(curplan))/t
      curplan ← newplan
    if (cost(curplan) < cost(best))
      best ← curplan
    reduce(t)
  return best
}
    
```

Annotations:

- Until "cooled down" (points to `t > 0`)
- Reduce Chance To "jump" (points to `reduce(t)`)
- Probability to Take "bad" plan Based on temp. (points to the exponential probability condition)

Genetic Algorithms

- Represent solutions as sequences (strings) = genome
- Start with random population of solutions
- Iterations = Generations
 - Mutation = random changes to genomes
 - Cross-over = Mixing two genomes

Genetic Join Enumeration for Left-deep Plans

- A left-deep plan can be represented as a permutation of the relations
 - Represent each relation by a number
 - E.g., encode this tree as "1243"



Mutation

- Switch random two random positions
- Is applied with a certain fixed probability
- E.g., "1342" -> "4312"

Cross-over

- Sub-set exchange
 - For two solutions find subsequence
 - equals length with the same set of relations
 - Exchange these subsequences
- Example
 - $J_1 = "5632478"$ and $J_2 = "5674328"$
 - Generate $J' = "5643278"$

Survival of the fittest

- Probability of survival determined by rank within the current population
- Compute ranks based on costs of solutions
- Assign Probabilities based on rank
 - Higher rank -> higher probability to survive
- Roll a dice for each solution

Genetic Join Enumeration

- Create an initial population P random plans
- Apply crossover and mutation with a fixed rate
 - E.g., crossover 65%, mutation 5%
- Apply selection until size is again P
- Stop once no improvement for at least X iterations

Comparison Randomized Join Enumeration

- Iterative Improvement
 - Towards local minima (easy to get stuck)
- Simulated Annealing
 - Probability to “jump” out of local minima
- Genetic Algorithms
 - Random transformation
 - Mixing solutions (crossover)
 - Probabilistic chance to keep solution based on cost

Join Enumeration Recap

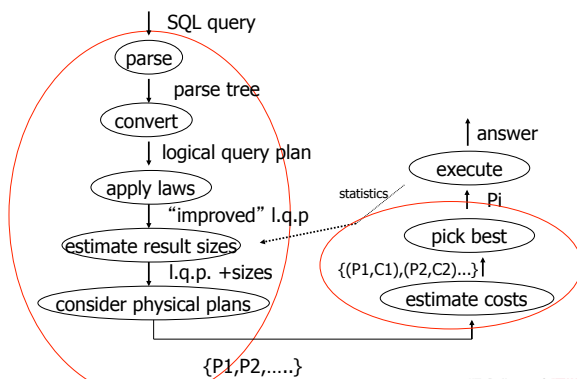
- Hard problem
 - Large problem size
 - Want to reduce search space
 - Large cost differences between solutions
 - Want to consider many solution to increase chance to find a good one.

Join Enumeration Recap

- Tip of the iceberg
 - More algorithms
 - Combinations of algorithms
 - Different representation subspaces of the problem
 - Cross-products / no cross-products
 - ...

From Join-Enumeration to Plan Enumeration

- So far we only know how to reorder joins
- What about other operations?
- What if the query does consist of several SQL blocks?
- What if we have nested subqueries?



From Join-Enumeration to Plan Enumeration

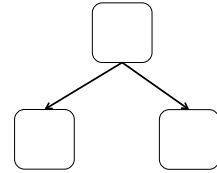
- Lets reconsider the input to plan enumeration!
 - We briefly touched on **Query graph models**
 - We discussed briefly why relational algebra is not sufficient

Query Graph Models

- Represents an SQL query as query blocks
 - A query block corresponds to the an SQL query block (SELECT FROM WHERE ...)
 - Data type/operator/function information
 - Needed for execution and optimization decisions
 - Structured in a way suited for optimization

QGM example

```
SELECT name, city
FROM
  (SELECT *
   FROM person) AS p,
  (SELECT *
   FROM address) AS a
WHERE p.addrId = a.id
```



Postgres Example

```
{QUERY
  commandType 1
  querySource 0
  scanRelTag true
  subRelID 0
  infoRelID 0
  infoClass 0
  pushAggs false
  pushSubLinks false
  cost 0
  STATE
  STATE
  (ALIAS
   relation p
   costname *)
  )
  (ALIAS
   relation p
   costname "name" "address")
  )
  subquery
  {QUERY
   commandType 1
   querySource 0
   scanRelTag true
  ...
```

How to enumerate plans for a QGM query

- Recall the correspondence between SQL query blocks and algebra expressions!
- If block is (A)SPJ
 - Determine join order
 - Decide which aggregation to use (if any)
- If block is set operation
 - Determine order

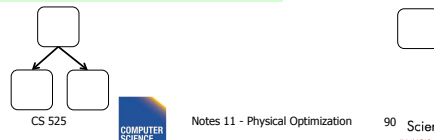
More than one query block

- Recursive create plans for subqueries
 - Start with leaf blocks
- Consider our example
 - Even if blocks are only SPJ we would not consider reordering of joins across blocks
 - -> try to "pull up" subqueries before optimization

Subquery Pull-up

```
SELECT name, city
FROM
  (SELECT *
   FROM person) AS p,
  (SELECT *
   FROM address) AS a
WHERE p.addrId = a.id
```

```
SELECT name, city
FROM
  person p,
  address a
WHERE p.addrId = a.id
```





Parameterized Queries

- Problem
 - Repeated executed of similar queries
- Example
 - Webshop
 - Typical operation: Retrieve product with all user comments for that product
 - Same query modulo product id

CS 525  Notes 11 - Physical Optimization 91  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Parameterized Queries

- Naïve approach
 - Optimize each version individually
 - Execute each version individually
- Materialized View
 - Store common parts of the query
 - -> Optimizing a query with materialized views
 - -> Separate topic not covered here

CS 525  Notes 11 - Physical Optimization 92  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY



Caching Query Plans

- Caching Query Plans
 - Optimize query once
 - Adapt plan for specific instances
 - **Assumption:** varying values do not effect optimization decisions
 - **Weaker Assumption:** Additional cost of "bad" plan less than cost of repeated planning

CS 525  Notes 11 - Physical Optimization 93  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Parameterized Queries

- How to represent varying parts of a query
 - Parameters
 - Query planned with parameters assumed to be unknown
 - For execution replace parameters with concrete values

CS 525  Notes 11 - Physical Optimization 94  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY



PREPARE statement

- In SQL
 - **PREPARE** name (parameters) AS query
 - **EXECUTE** name (parameters)

CS 525  Notes 11 - Physical Optimization 95  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Nested Subqueries

```
SELECT name
FROM person p
WHERE EXISTS (SELECT newspaper
              FROM hasRead h
              WHERE h.name = p.name
                 AND h.newspaper = 'Tribune')
```

CS 525  Notes 11 - Physical Optimization 96  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

How to evaluate nested subquery?

- If no correlations:
 - Execute once and cache results
- For correlations:
 - Create plan for query with parameters
- -> called nested iteration

Nested Iteration - Correlated

```
q ← outer query
q' ← inner query
result ← execute(q)
foreach tuple t in result
  qt ← q'(t) // parameterize q' with values from t
  result' ← execute (qt)
  evaluate_nested_condition (t,result')
```

Nested Iteration - Uncorrelated

```
q ← outer query
q' ← inner query
result ← execute(q)
result' ← execute (qt)
foreach tuple t in result
  evaluate_nested_condition (t,result')
```

Nested Iteration - Example

```
SELECT name
FROM person p
WHERE EXISTS (SELECT newspaper
              FROM hasRead h
              WHERE h.name = p.name
              AND h.newspaper = 'Tribune')
```

person		hasRead	
name	gender	name	newspaper
Alice	female	Alice	Tribune
Bob	male	Alice	Courier
Joe	male	Joe	Courier

Nested Iteration - Example

```
q ← outer query
q' ← inner query
result ← execute(q)
foreach tuple t in result
  qt ← q'(t)
  result' ← execute (qt)
  evaluate_nested_condition (t,result')
```

```
SELECT newspaper
FROM hasRead h
WHERE h.name = p.name
AND h.newspaper = 'Tribune'
```

person		hasRead	
name	gender	name	newspaper
→ Alice	female	Alice	Tribune
Bob	male	Alice	Courier
Joe	male	Joe	Courier

Nested Iteration - Example

```
q ← outer query
q' ← inner query
result ← execute(q)
foreach tuple t in result
  qt ← q'(t)
  result' ← execute (qt)
  evaluate_nested_condition (t,result')
```

```
SELECT newspaper
FROM hasRead h
WHERE h.name = 'Alice'
AND h.newspaper = 'Tribune'
```

person		hasRead	
name	gender	name	newspaper
→ Alice	female	Alice	Tribune
Bob	male	Alice	Courier
Joe	male	Joe	Courier

Nested Iteration - Example

```

q - outer query
q' - inner query
result - execute(q)
foreach tuple t in result
  q_t - q'(t)
  result' - execute(q_t)
  evaluate_nested_condition(t, result')
    
```

```

SELECT newspaper
FROM hasRead h
WHERE h.name = p.name
      AND h.newspaper
      = 'Tribune'
    
```

person		hasRead		result'
name	gender	name	newspaper	newspaper
Alice	female	Alice	Tribune	Tribune
Bob	male	Alice	Courier	
Joe	male	Joe	Courier	

CS 525



Notes 11 - Physical Optimization

103

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Nested Iteration - Example

```

q - outer query
q' - inner query
result - execute(q)
foreach tuple t in result
  q_t - q'(t)
  result' - execute(q_t)
  evaluate_nested_condition(t, result')
    
```

EXISTS evaluates to true!

Output(Alice)

person		hasRead		result'
name	gender	name	newspaper	newspaper
Alice	female	Alice	Tribune	Tribune
Bob	male	Alice	Courier	
Joe	male	Joe	Courier	

CS 525



Notes 11 - Physical Optimization

104

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Nested Iteration - Example

```

q - outer query
q' - inner query
result - execute(q)
foreach tuple t in result
  q_t - q'(t)
  result' - execute(q_t)
  evaluate_nested_condition(t, result')
    
```

Empty result set ->
EXISTS evaluates to false

person		hasRead		result'
name	gender	name	newspaper	newspaper
Alice	female	Alice	Tribune	
Bob	male	Alice	Courier	
Joe	male	Joe	Courier	

CS 525



Notes 11 - Physical Optimization

105

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Nested Iteration - Example

```

q - outer query
q' - inner query
result - execute(q)
foreach tuple t in result
  q_t - q'(t)
  result' - execute(q_t)
  evaluate_nested_condition(t, result')
    
```

Empty result set ->
EXISTS evaluates to false

person		hasRead		result'
name	gender	name	newspaper	newspaper
Alice	female	Alice	Tribune	
Bob	male	Alice	Courier	
Joe	male	Joe	Courier	

CS 525



Notes 11 - Physical Optimization

106

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Nested Iteration - Discussion

- Repeated evaluation of nested subquery
 - If correlated
 - Improve:
 - Plan once and substitute parameters
 - EXISTS: stop processing after first result
 - IN/ANY: stop after first match
- No optimization across nesting boundaries

CS 525



Notes 11 - Physical Optimization

107

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Unnesting and Decorrelation

- Apply equivalences to transform nested subqueries into joins
- **Unnesting:**
 - Turn a nested subquery into a join
- **Decorrelation:**
 - Turn correlations into join expressions

CS 525



Notes 11 - Physical Optimization

108

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Equivalences

- Classify types of nesting
- Equivalence rules will have preconditions
- Can be applied heuristically before plan enumeration or using a transformative approach



CS 525  Notes 11 - Physical Optimization 109  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

N-type Nesting

- Properties
 - Expression ANY comparison (or IN)
 - No Correlations
 - Nested query does not use aggregation

• Example

```
SELECT name
FROM orders o
WHERE o.cust IN (SELECT cId
                 FROM customer
                 WHERE region = 'USA')
```



CS 525  Notes 11 - Physical Optimization 110  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

A-type Nesting

- Properties
 - Expression is ANY comparison (or scalar)
 - No Correlations
 - Nested query uses aggregation
 - No Group By

• Example

```
SELECT name
FROM orders o
WHERE o.amount = (SELECT max(amount)
                  FROM orders i)
```



CS 525  Notes 11 - Physical Optimization 111  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

J-type Nesting

- Properties
 - Expression is ANY comparison (IN)
 - Nested query uses equality comparison with correlated attribute
 - No aggregation in nested query

• Example

```
SELECT name
FROM orders o
WHERE o.amount IN (SELECT amount
                   FROM orders i
                   WHERE i.cust = o.cust
                   AND i.shop = 'New York')
```



CS 525  Notes 11 - Physical Optimization 114  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

JA-type Nesting

- Properties
 - Expression equality comparison
 - Nested query uses equality comparison with correlated attribute
 - Nested query uses aggregation and no GROUP BY



• Example

```
SELECT name
FROM orders o
WHERE o.amount = (SELECT max(amount)
                  FROM orders i
                  WHERE i.cust = o.cust)
```

CS 525  Notes 11 - Physical Optimization 114  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Unnesting A-type

- Move nested query to FROM clause
- Turn nested condition (op ANY, IN) into op with result attribute of nested query

CS 525  Notes 11 - Physical Optimization 114  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Unnesting N/J-type

- Move nested query to FROM clause
- Add DISTINCT to SELECT clause of nested query
- Turn equality comparison with correlated attributes into join conditions
- Turn nested condition (op ANY, IN) into op with result attribute of nested query

Example

1. To FROM clause
2. Add DISTINCT
3. Correlation to join
4. Nesting condition to join

```
SELECT name
FROM orders o
WHERE o.amount IN (SELECT amount
                   FROM orders i
                   WHERE i.cust = o.cust
                   AND i.shop = 'New York')
```

```
SELECT name
FROM orders o,
(SELECT amount
 FROM orders i
 WHERE i.cust = o.cust
 AND i.shop = 'New York') AS sub
```

Example

1. To FROM clause
2. Add DISTINCT
3. Correlation to join
4. Nesting condition to join

```
SELECT name
FROM orders o
WHERE o.amount IN (SELECT amount
                   FROM orders i
                   WHERE i.cust = o.cust
                   AND i.shop = 'New York')
```

```
SELECT name
FROM orders o,
(SELECT DISTINCT amount
 FROM orders i
 WHERE i.cust = o.cust
 AND i.shop = 'New York') AS sub
```

Example

1. To FROM clause
2. Add DISTINCT
3. Correlation to join
4. Nesting condition to join

```
SELECT name
FROM orders o
WHERE o.amount IN (SELECT amount
                   FROM orders i
                   WHERE i.cust = o.cust
                   AND i.shop = 'New York')
```

```
SELECT name
FROM orders o,
(SELECT DISTINCT amount, cust
 FROM orders i
 WHERE i.shop = 'New York') AS sub
WHERE sub.cust = o.cust
```

Example

1. To FROM clause
2. Add DISTINCT
3. Correlation to join
4. Nesting condition to join

```
SELECT name
FROM orders o
WHERE o.amount IN (SELECT amount
                   FROM orders i
                   WHERE i.cust = o.cust
                   AND i.shop = 'New York')
```

```
SELECT name
FROM orders o,
(SELECT DISTINCT amount, cust
 FROM orders i
 WHERE i.shop = 'New York') AS sub
WHERE sub.cust = o.cust
AND o.amount = sub.amount
```

Unnesting JA-type

- Move nested query to FROM clause
- Turn equality comparison with correlated attributes into
 - GROUP BY
 - Join conditions
- Turn nested condition (op ANY, IN) into op with result attribute of nested query

Example

1. To FROM clause
2. Introduce GROUP BY and join conditions
3. Nesting condition to join

```
SELECT name
FROM orders o
WHERE o.amount = (SELECT max(amount)
                  FROM orders i
                  WHERE i.cust = o.cust)
```

```
SELECT name
FROM orders o,
     (SELECT max(amount)
      FROM orders I
      WHERE i.cust = o.cust) sub
```

CS 525



Notes 11 - Physical Optimization

121

Example

1. To FROM clause
2. Introduce GROUP BY and join conditions
3. Nesting condition to join

```
SELECT name
FROM orders o
WHERE o.amount = (SELECT max(amount)
                  FROM orders i
                  WHERE i.cust = o.cust)
```

```
SELECT name
FROM orders o,
     (SELECT max(amount) AS ma, i.cust
      FROM orders i
      GROUP BY i.cust) sub
WHERE i.cust = sub.cust
```

CS 525



Notes 11 - Physical Optimization

122

Example

1. To FROM clause
2. Introduce GROUP BY and join conditions
3. Nesting condition to join

```
SELECT name
FROM orders o
WHERE o.amount = (SELECT max(amount)
                  FROM orders i
                  WHERE i.cust = o.cust)
```

```
SELECT name
FROM orders o,
     (SELECT max(amount) AS ma, i.cust
      FROM orders i
      GROUP BY i.cust) sub
WHERE sub.cust = o.cust
      AND o.amount = sub.ma
```

CS 525



Notes 11 - Physical Optimization

123

Unnesting Benefits Example

- $N(\text{orders}) = 1,000,000$
- $V(\text{cust,orders}) = 10,000$
- $S(\text{orders}) = 1/10$ block

```
SELECT name
FROM orders o
WHERE o.amount = (SELECT max(amount)
                  FROM orders i
                  WHERE i.cust = o.cust)
```

```
SELECT name
FROM orders o,
     (SELECT max(amount) AS ma, i.cust
      FROM orders i
      GROUP BY i.cust) sub
WHERE sub.cust = o.cust
      AND o.amount = sub.ma
```

CS 525



Notes 11 - Physical Optimization

124

- $N(\text{orders}) = 1,000,000$
- $V(\text{cust,orders}) = 10,000$
- $S(\text{orders}) = 1/10$ block
- $M = 10,000$

```
SELECT name
FROM orders o
WHERE o.amount = (SELECT max(amount)
                  FROM orders i
                  WHERE i.cust = o.cust)
```

- Inner query:
 - One scan $B(\text{orders}) = 100,000$ I/Os
- Outer query:
 - One scan $B(\text{orders}) = 100,000$ I/Os
 - 1,000,000 tuples
- Total cost: $1,000,001 \times 100,000 \approx 10^{11}$ I/Os

CS 525



Notes 11 - Physical Optimization

125

- $N(\text{orders}) = 1,000,000$
- $V(\text{cust,orders}) = 10,000$
- $S(\text{orders}) = 1/10$ block
- $M = 10,000$

```
SELECT name
FROM orders o,
     (SELECT max(amount) AS ma, i.cust
      FROM orders i
      GROUP BY i.cust) sub
WHERE sub.cust = o.cust
      AND o.amount = sub.ma
```

- Inner queries:
 - One scan $B(\text{orders}) = 100,000$ I/Os
 - 1,000,000 result tuples
 - Aggregation: Sort (assume 1 pass) = $3 \times 100,000 = 300,000$ I/Os
 - 10,000 result tuples $\rightarrow + 1,000$ pages to write to disk
- The join: use merge – join during merge
 - $3 \times (1,000 + 100,000)$ I/Os = 303,000 I/Os
- Total cost: 604,000 I/Os

CS 525




Notes 11 - Physical Optimization

126



CS 525: Advanced Database Organization

12: Transaction Management



Boris Glavic



Slides: adapted from a [course](#) taught by [Hector Garcia-Molina](#), Stanford InfoLab

CS 525  Notes 12 - Transaction Management 1  IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

1

Concurrency and Recovery



- DBMS should enable multiple clients to access the database concurrently
 - This can lead to problems with correctness of data because of interleaving of operations from different clients
 - ->System should ensure correctness (**concurrency control**)

CS 525  Notes 12 - Transaction Management 2  IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

2

Concurrency and Recovery

- DBMS should enable reestablish correctness of data in the presence of failures
 - ->System should restore a correct state after failure (**recovery**)

CS 525  Notes 12 - Transaction Management 3  IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY



3

Integrity or correctness of data

- Would like data to be “accurate” or “correct” at all times

EMP



Name	Age
White	52
Green	3421
Gray	1

CS 525  Notes 12 - Transaction Management 4  IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

4

Integrity or consistency constraints



- Predicates data must satisfy
- Examples:
 - x is key of relation R
 - $x \rightarrow y$ holds in R
 - Domain(x) = {Red, Blue, Green}
 - α is valid index for attribute x of R
 - no employee should make more than twice the average salary

CS 525  Notes 12 - Transaction Management 5  IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

5

Definition:

- Consistent state: satisfies all constraints
- Consistent DB: DB in consistent state

CS 525  Notes 12 - Transaction Management 6  IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

6

Constraints (as we use here) may not capture “full correctness”

Example 1 Transaction constraints

- When salary is updated, new salary > old salary
- When account record is deleted, balance = 0

7

Note: could be “emulated” by simple constraints, e.g.,

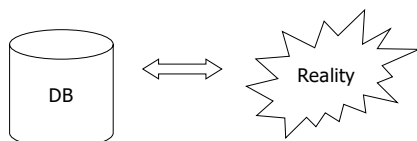
account

Acct #	...	balance	deleted?
--------	-----	---------	----------

8

Constraints (as we use here) may not capture “full correctness”

Example 2 Database should reflect real world



9

in any case, continue with constraints...

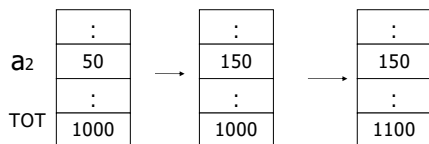
Observation: DB cannot be consistent always!

Example: $a_1 + a_2 + \dots + a_n = \text{TOT}$ (constraint)
 Deposit \$100 in a_2 : $\begin{cases} a_2 \leftarrow a_2 + 100 \\ \text{TOT} \leftarrow \text{TOT} + 100 \end{cases}$

10

Example: $a_1 + a_2 + \dots + a_n = \text{TOT}$ (constraint)

Deposit \$100 in a_2 : $\begin{cases} a_2 \leftarrow a_2 + 100 \\ \text{TOT} \leftarrow \text{TOT} + 100 \end{cases}$



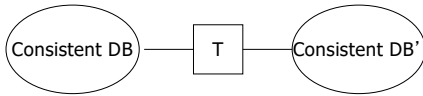
11

Transactions

- **Transaction:** Sequence of operations executed by one concurrent client that preserve consistency

12

Transaction: collection of actions that preserve consistency



CS 525



Notes 12 - Transaction Management

13

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

13

Big assumption:

If T starts with consistent state +
T executes in isolation
⇒ T leaves consistent state

CS 525



Notes 12 - Transaction Management

14

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

14

Correctness (informally)

- If we stop running transactions, DB left consistent
- Each transaction sees a consistent DB

CS 525



Notes 12 - Transaction Management

15

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

15

Transactions - ACID

- **Atomicity**
 - Either all or no commands of transaction are executed (their changes are persisted in the DB)
- **Consistency**
 - After transaction DB is consistent (if before consistent)
- **Isolation**
 - Transactions are running isolated from each other
- **Durability**
 - Modifications of transactions are never lost

CS 525



Notes 12 - Transaction Management

16

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

16

How can constraints be violated?

- Transaction bug
- DBMS bug
- Hardware failure
 - e.g., disk crash alters balance of account
- Data sharing
 - e.g.: T1: give 10% raise to programmers
 - T2: change programmers ⇒ systems analysts

CS 525



Notes 12 - Transaction Management

17

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

17

How can we prevent/fix violations?

- Part 13 (Recovery):
 - due to failures
- Part 14 (Concurrency Control):
 - due to data sharing

CS 525



Notes 12 - Transaction Management

18

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

18

Will not consider:

- How to write correct transactions
 - How to write correct DBMS
 - Constraint checking & repair
- That is, solutions studied here do not need to know constraints

CS 525



Notes 12 - Transaction Management

19



19

Data Items:

- **Data Item / Database Object / ...**
- Abstraction that will come in handy when talking about concurrency control and recovery
- Data Item could be
 - Table, Row, Page, Attribute value

CS 525



Notes 12 - Transaction Management

20



20

Operations:

- Input (x): block containing x → memory
- Output (x): block containing x → disk

CS 525



Notes 12 - Transaction Management

21



21

Operations:

- Input (x): block containing x → memory
- Output (x): block containing x → disk
- Read (x,t): do input(x) if necessary
t ← value of x in block
- Write (x,t): do input(x) if necessary
value of x in block ← t

CS 525



Notes 12 - Transaction Management

22



22

Key problem Unfinished transaction
(Atomicity)

Example Constraint: A=B
T₁: A ← A × 2
B ← B × 2

CS 525



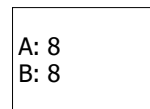
Notes 12 - Transaction Management

23

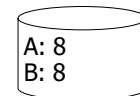


23

T₁: Read (A,t); t ← t×2
Write (A,t);
Read (B,t); t ← t×2
Write (B,t);
Output (A);
Output (B);



memory



disk

CS 525



Notes 12 - Transaction Management

24



24

T1: Read (A,t); t ← t×2
 Write (A,t);
 Read (B,t); t ← t×2
 Write (B,t);
 Output (A);
 Output (B);

A: ~~8~~ 16
 B: ~~8~~ 16

A: 8
 B: 8

memory disk

CS 525 Notes 12 - Transaction Management 25 IIT College of Science and Letters
 ILLINOIS INSTITUTE OF TECHNOLOGY

25

T1: Read (A,t); t ← t×2
 Write (A,t);
 Read (B,t); t ← t×2
 Write (B,t);
 Output (A);
 Output (B);

failure!

A: ~~8~~ 16
 B: ~~8~~ 16

A: ~~8~~ 16
 B: 8

memory disk

CS 525 Notes 12 - Transaction Management 26 IIT College of Science and Letters
 ILLINOIS INSTITUTE OF TECHNOLOGY

26

Transactions in SQL

- **BEGIN WORK**
 - Start new transaction
 - Often implicit
- **COMMIT**
 - Finish and make all modifications of transactions persistent
- **ABORT/ROLLBACK**
 - Finish and undo all changes of transaction

CS 525 Notes 12 - Transaction Management 27 IIT College of Science and Letters
 ILLINOIS INSTITUTE OF TECHNOLOGY

27

Example

time ↓

BEGIN WORK;
 UPDATE accounts
 SET bal = bal + 40
 WHERE acc = 10;

BEGIN WORK;
 UPDATE accounts
 SET bal = bal * 1.05;
 COMMIT;

UPDATE accounts
 SET bal = bal - 40
 WHERE acc = 9;
 COMMIT;

CS 525 Notes 12 - Transaction Management 28 IIT College of Science and Letters
 ILLINOIS INSTITUTE OF TECHNOLOGY

28

Example

time ↓

BEGIN WORK;
 UPDATE accounts
 SET bal = bal + 40
 WHERE acc = 10;

BEGIN WORK;
 UPDATE accounts
 SET bal = bal * 1.05;
 COMMIT;

UPDATE accounts
 SET bal = bal - 40
 WHERE acc = 9;
 COMMIT;

Bank customer transfers money from account 9 to account 10

CS 525 Notes 12 - Transaction Management 29 IIT College of Science and Letters
 ILLINOIS INSTITUTE OF TECHNOLOGY

29

Example

time ↓

BEGIN WORK;
 UPDATE accounts
 SET bal = bal + 40
 WHERE acc = 10;

BEGIN WORK;
 UPDATE accounts
 SET bal = bal * 1.05;
 COMMIT;

UPDATE accounts
 SET bal = bal - 40
 WHERE acc = 9;
 COMMIT;

Bank adds interest to all accounts

CS 525 Notes 12 - Transaction Management 30 IIT College of Science and Letters
 ILLINOIS INSTITUTE OF TECHNOLOGY

30

time

```

BEGIN WORK;
UPDATE accounts
SET bal = bal
WHERE acc = 10

UPDATE accounts
SET bal = bal - 40
WHERE acc = 9;
COMMIT;

```



Potential Problems:

- Transactions are interrupted
 - No reduction in bal of acc 9
 - Only some accounts got interest
- Interleaving of Transaction
 - Acc 9 too much interest (before 40 has been deducted)

```

SET bal = bal * 1.05;
COMMIT;



```

CS 525  Notes 12 - Transaction Management 31  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

31

Modeling Transactions and their Interleaving

- Transaction is sequence of operations
 - read:** $r_i(x)$ = transaction i read item x
 - write:** $w_i(x)$ = transaction i wrote item x
 - commit:** c_i = transaction i committed
 - abort:** a_i = transaction i aborted

CS 525  Notes 12 - Transaction Management 32  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

32

$T_1 = r_1(a_{10}), w_1(a_{10}), r_1(a_9), w_1(a_9), c_1$



time

```

BEGIN WORK;
UPDATE accounts
SET bal = bal + 40
WHERE acc = 10;

UPDATE accounts
SET bal = bal - 40
WHERE acc = 9;
COMMIT;

```

CS 525  Notes 12 - Transaction Management 33  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

33

$T_1 = r_1(a_{10}), w_1(a_{10}), r_1(a_9), w_1(a_9), c_1$

$T_2 = r_2(a_1), w_2(a_1), r_2(a_2), w_2(a_2), r_2(a_9), w_2(a_9), r_2(a_{10}), w_2(a_{10}), c_2$

time

```

BEGIN WORK;
UPDATE accounts
SET bal = bal + 40
WHERE acc = 10;

UPDATE accounts
SET bal = bal - 40
WHERE acc = 9;
COMMIT;



```

Assume we have accounts:
 a_1, a_2, a_9, a_{10}

```

BEGIN WORK;
UPDATE accounts
SET bal = bal * 1.05;
COMMIT;



```

CS 525  Notes 12 - Transaction Management 34  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

34

Schedules



- A **schedule S** for a set of transactions $T = \{T_1, \dots, T_n\}$ is an partial order over operations of T so that
 - S** contains a prefix of the operations of each T_i
 - Operations of T_i appear in the same order in **S** as in T_i
 - For any two conflicting operations they are ordered

CS 525  Notes 12 - Transaction Management 35  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

35

Note

- For simplicity: We often assume that the schedule is a total order

CS 525  Notes 12 - Transaction Management 36  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

36

How to model execution order?

- Schedules model the order of the execution for operations of a set of transactions

CS 525
Notes 12 - Transaction Management
37

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

37

Conflicting Operations

- Two operations are conflicting if
 - At least one of them is a write
 - Both are accessing the same data item
- Intuition
 - The order of execution for conflicting operations can influence result!

CS 525
Notes 12 - Transaction Management
38

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

38

Conflicting Operations

- Examples
 - $w_1(X), r_2(X)$ are conflicting
 - $w_1(X), w_2(Y)$ are not conflicting
 - $r_1(X), r_2(X)$ are not conflicting
 - $w_1(X), w_1(X)$ are not conflicting

CS 525
Notes 12 - Transaction Management
39

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

39

Complete Schedules = History

- A **schedule S** for T is complete if it contains all operations from each transaction in T
- We will call complete schedules **histories**

CS 525
Notes 12 - Transaction Management
40

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

40

$T_1 = r_1(a_{10}), w_1(a_{10}), r_1(a_9), w_1(a_9), c_1$

$T_2 = r_2(a_1), w_2(a_1), r_2(a_2), w_2(a_2), r_2(a_9), w_2(a_9), r_2(a_{10}), w_2(a_{10}), c_1$

Complete Schedule

$S = r_2(a_1), r_1(a_{10}), w_2(a_1), r_2(a_2), w_1(a_{10}), w_2(a_2), r_2(a_9), w_2(a_9), r_1(a_9), w_1(a_9), c_1, r_2(a_{10}), w_2(a_{10}), c_1$

Incomplete Schedule

$S = r_2(a_1), r_1(a_{10}), w_2(a_1), w_1(a_{10})$

Not a Schedule

$S = r_2(a_1), r_1(a_{10}), c_1$

CS 525
Notes 12 - Transaction Management
41

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

41

$T_1 = r_1(a_{10}), w_1(a_{10}), r_1(a_9), w_1(a_9), c_1$

$T_2 = r_2(a_1), w_2(a_1), r_2(a_2), w_2(a_2), r_2(a_9), w_2(a_9), r_2(a_{10}), w_2(a_{10}), c_2$

Conflicting operations

- Conflicting operations $w_1(a_{10})$ and $w_2(a_{10})$
- Order of these operations determines value of a_{10}
- S1 and S2 do not generate the same result

$S_1 = \dots w_2(a_{10}) \dots w_1(a_{10})$

$S_2 = \dots w_1(a_{10}) \dots w_2(a_{10})$

CS 525
Notes 12 - Transaction Management
42

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

42

Why Schedules?

- Study properties of different execution orders
 - Easy/Possible to recover after failure
 - Isolation
 - -> preserve ACID properties
- Classes of schedules and protocols to guarantee that only "good" schedules are produced

CS 525



Notes 12 - Transaction Management

43

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

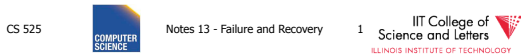


CS 525: Advanced Database Organization

13: Failure and Recovery

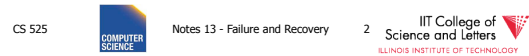
Boris Glavic

Slides: adapted from a [course](#) taught by [Hector Garcia-Molina](#), Stanford InfoLab



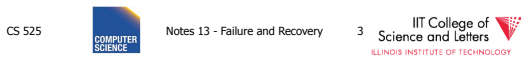
Now

• Crash recovery



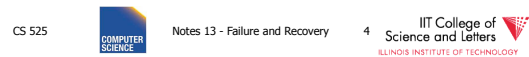
Correctness (informally)

- If we stop running transactions, DB left consistent
- Each transaction sees a consistent DB



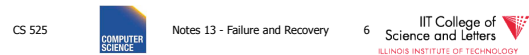
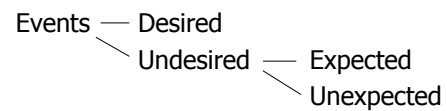
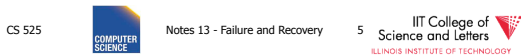
How can constraints be violated?

- Transaction bug
- DBMS bug
- Hardware failure
 - e.g., disk crash alters balance of account
- Data sharing
 - e.g.: T1: give 10% raise to programmers
 - T2: change programmers ⇒ systems analysts

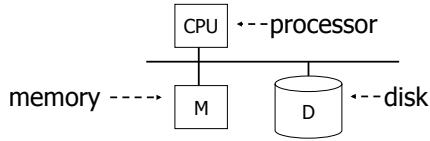


Recovery

- First order of business: Failure Model



Our failure model



Desired events: see product manuals....

Undesired expected events:
System crash
- memory lost
- cpu halts, resets

Desired events: see product manuals....

Undesired expected events:
System crash
- memory lost
- cpu halts, resets

----- that's it!! -----

Undesired Unexpected: Everything else!

Undesired Unexpected: Everything else!

Examples:
• Disk data is lost
• Memory lost without CPU halt
• CPU implodes wiping out universe....

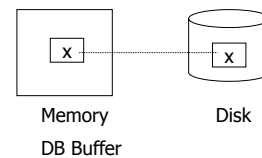
Is this model reasonable?

Approach: Add low level checks + redundancy to increase probability model holds

E.g., { Replicate disk storage (stable store)
Memory parity
CPU checks

Second order of business:

Storage hierarchy



Operations:

- Input (x): block containing x → memory
- Output (x): block containing x → disk

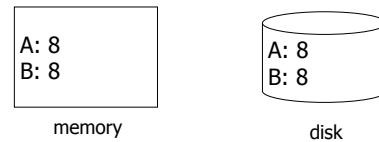
Operations:

- Input (x): block containing x → memory
- Output (x): block containing x → disk
- Read (x,t): do input(x) if necessary
t ← value of x in block
- Write (x,t): do input(x) if necessary
value of x in block ← t

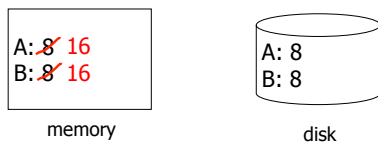
Key problem Unfinished transaction

Example Constraint: A=B
 T1: A ← A × 2
 B ← B × 2

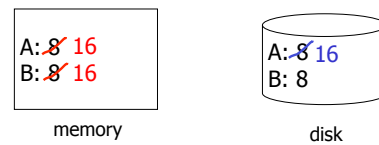
T1: Read (A,t); t ← t×2
 Write (A,t);
 Read (B,t); t ← t×2
 Write (B,t);
 Output (A);
 Output (B);



T1: Read (A,t); t ← t×2
 Write (A,t);
 Read (B,t); t ← t×2
 Write (B,t);
 Output (A);
 Output (B);



T1: Read (A,t); t ← t×2
 Write (A,t);
 Read (B,t); t ← t×2
 Write (B,t);
 Output (A);
 Output (B); failure!



- Need atomicity:
 - execute all actions of a transaction or none at all

How to restore consistent state after crash?

- Desired state after recovery:
 - Changes of committed transactions are reflected on disk
 - Changes of unfinished transactions are not reflected on disk
- After crash we need to
 - **Undo** changes of unfinished transactions that have been written to disk
 - **Redo** changes of finished transactions that have not been written to disk

How to restore consistent state after crash?

- After crash we need to
 - **Undo** changes of unfinished transactions that have been written to disk
 - **Redo** changes of finished transactions that have not been written to disk
- We need to either
 - Store additional data to be able to Undo/Redo
 - Avoid ending up in situations where we need to Undo/Redo

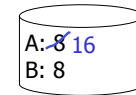
T₁: Read (A,t); t ← tx2
 Write (A,t);
 Read (B,t); t ← tx2
 Write (B,t);
Output (A);
 Output (B);

T₁ is unfinished
 -> need to undo the write to A to recover to consistent state

failure!



memory



disk

Logging

- After crash need to
 - **Undo**
 - **Redo**
- We need to know
 - Which operations have been executed
 - Which operations are reflected on disk
- -> **Log** upfront what is to be done

Buffer Replacement Revisited

- Now we are interested in knowing how buffer replacement influences recovery!

Buffer Replacement Revisited

- **Steal:** all pages with fix count = 0 are replacement candidates
 - Smaller buffer requirements
- **No steal:** pages that have been modified by active transaction -> not considered for replacement
 - No need to undo operations of unfinished transactions after failure

Buffer Replacement Revisited

- **Force:** Pages modified by transaction are flushed to disk at end of transaction
 - No redo required
- **No force:** modified (dirty) pages are allowed to remain in buffer after end of transaction
 - Less repeated writes of same page

Effects of Buffer Replacement

	force	No force
No steal	<ul style="list-style-type: none"> • No Undo • No Redo 	<ul style="list-style-type: none"> • No Undo • Redo
steal	<ul style="list-style-type: none"> • Undo • No Redo 	<ul style="list-style-type: none"> • Redo • Undo

Schedules and Recovery

- Are there certain schedules that are easy/hard/impossible to recover from?

Recoverable Schedules

- We should never have to rollback an already committed transaction (D in ACID)
- **Recoverable (RC)** schedules require that
 - A transaction does not commit before every transaction that is has read from has committed
 - A transaction **T** reads from another transaction **T'** if it reads an item X that has last been written by T' and T' has not aborted before the read

$$T_1 = w_1(X), c_1$$

$$T_2 = r_2(X), w_2(X), c_2$$

Recoverable (RC) Schedule

$$S_1 = w_1(X), r_2(X), w_2(X), c_1, c_2$$

Nonrecoverable Schedule

$$S_2 = w_1(X), r_2(X), w_2(X), c_2, c_1$$

Cascading Abort

- Transaction **T** has written an item that is later read by **T'** and **T** aborts after that
 - we have to also abort **T'** because the value it read is no longer valid anymore
 - This is called a **cascading abort**
 - Cascading aborts are complex and should be avoided

$$S = \dots w_1(X) \dots r_2(X) \dots a_1$$

Cascadeless Schedules

- Cascadeless (CL)** schedules guarantee that there are no cascading aborts
 - Transactions only read values written by already committed transactions

$$T_1 = w_1(X), c_1$$

$$T_2 = r_2(X), w_2(X), c_2$$

Cascadeless (CL) Schedule

$$S_1 = w_1(X), c_1, r_2(X), w_2(X), c_2$$

Recoverable (RC) Schedule

$$S_2 = w_1(X), r_2(X), w_2(X), c_1, c_2$$

Nonrecoverable Schedule

$$S_3 = w_1(X), r_2(X), w_2(X), c_2, c_1$$

$$T_1 = w_1(X), a_1$$

$$T_2 = r_2(X), w_2(X), c_2$$

Cascadeless (CL) Schedule

$$S_1 = w_1(X), a_1, r_2(X), w_2(X), c_2$$

Recoverable (RC) Schedule

$$S_2 = w_1(X), r_2(X), w_2(X), a_1, a_2$$

Nonrecoverable Schedule

$$S_3 = w_1(X), r_2(X), w_2(X), c_2, a_1$$

Consider what happens if T1 aborts!

Strict Schedules

- Strict (ST)** schedules guarantee that to Undo the effect of a transaction we simply have to undo each of its writes
 - Transactions do not read nor write items written by uncommitted transactions

$$T_1 = w_1(X), c_1$$

$$T_2 = r_2(X), w_2(X), c_2$$

Cascadeless (CL) + Strict Schedule (ST)

$$S_1 = w_1(X), c_1, r_2(X), w_2(X), c_2$$

Recoverable (RC) Schedule

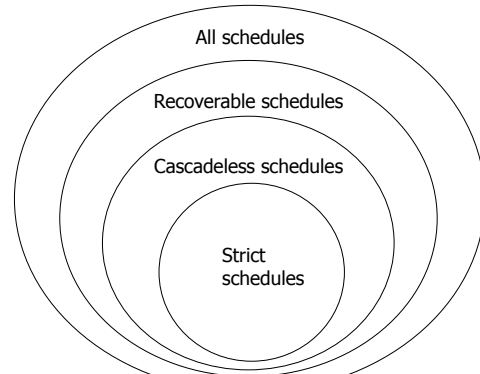
$$S_2 = w_1(X), r_2(X), w_2(X), c_1, c_2$$

Nonrecoverable Schedule

$$S_3 = w_1(X), r_2(X), w_2(X), c_2, c_1$$

Compare Classes

ST \subset CL \subset RC \subset ALL



Logging and Recovery

- We now discuss approaches for logging and how to use them in recovery

One solution: undo logging (immediate modification)

due to: Hansel and Gretel, 782 AD

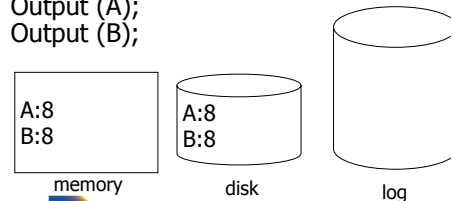
One solution: undo logging (immediate modification)

due to: Hansel and Gretel, 782 AD

- Improved in 784 AD to durable undo logging

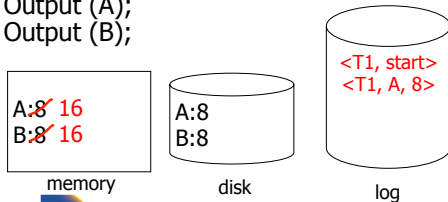
Undo logging (Immediate modification)

T1: Read (A,t); $t \leftarrow tx2$ A=B
Write (A,t);
Read (B,t); $t \leftarrow tx2$
Write (B,t);
Output (A);
Output (B);



Undo logging (Immediate modification)

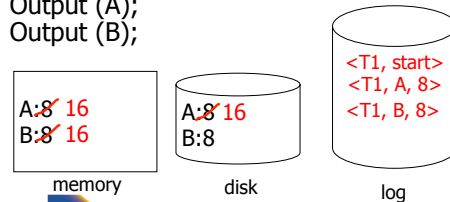
T1: Read (A,t); $t \leftarrow t \times 2$ A=B
 Write (A,t);
 Read (B,t); $t \leftarrow t \times 2$
 Write (B,t);
 Output (A);
 Output (B);



CS 525 Notes 13 - Failure and Recovery 43 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

Undo logging (Immediate modification)

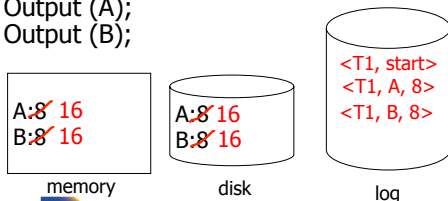
T1: Read (A,t); $t \leftarrow t \times 2$ A=B
 Write (A,t);
 Read (B,t); $t \leftarrow t \times 2$
 Write (B,t);
 Output (A);
 Output (B);



CS 525 Notes 13 - Failure and Recovery 44 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

Undo logging (Immediate modification)

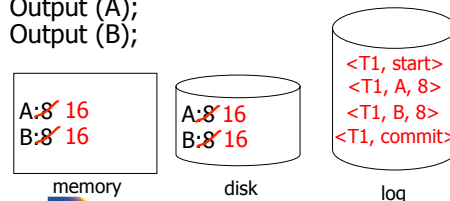
T1: Read (A,t); $t \leftarrow t \times 2$ A=B
 Write (A,t);
 Read (B,t); $t \leftarrow t \times 2$
 Write (B,t);
 Output (A);
 Output (B);



CS 525 Notes 13 - Failure and Recovery 45 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

Undo logging (Immediate modification)

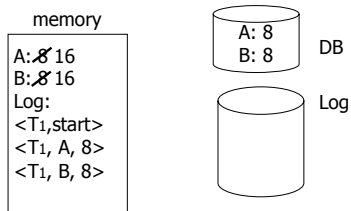
T1: Read (A,t); $t \leftarrow t \times 2$ A=B
 Write (A,t);
 Read (B,t); $t \leftarrow t \times 2$
 Write (B,t);
 Output (A);
 Output (B);



CS 525 Notes 13 - Failure and Recovery 46 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

One “complication”

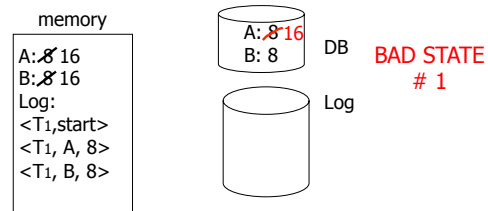
- Log is first written in memory
- Not written to disk on every action



CS 525 Notes 13 - Failure and Recovery 47 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

One “complication”

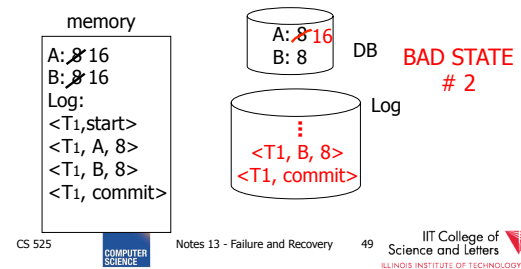
- Log is first written in memory
- Not written to disk on every action



CS 525 Notes 13 - Failure and Recovery 48 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

One “complication”

- Log is first written in memory
- Not written to disk on every action



Undo logging rules

- (1) For every action generate undo log record (containing old value)
- (2) Before x is modified on disk, log records pertaining to x must be on disk (write ahead logging: **WAL**)
- (3) Before commit is flushed to log, all writes of transaction must be reflected on disk

Recovery rules: Undo logging

- For every T_i with $\langle T_i, \text{start} \rangle$ in log:
 - If $\langle T_i, \text{commit} \rangle$ or $\langle T_i, \text{abort} \rangle$ in log, do nothing
 - Else { For all $\langle T_i, X, v \rangle$ in log:
 - write (X, v)
 - output (X)
 Write $\langle T_i, \text{abort} \rangle$ to log

Recovery rules: Undo logging

- For every T_i with $\langle T_i, \text{start} \rangle$ in log:
 - If $\langle T_i, \text{commit} \rangle$ or $\langle T_i, \text{abort} \rangle$ in log, do nothing
 - Else { For all $\langle T_i, X, v \rangle$ in log:
 - write (X, v)
 - output (X)
 Write $\langle T_i, \text{abort} \rangle$ to log

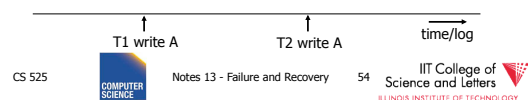
IS THIS CORRECT??

Recovery rules: Undo logging

- (1) Let S = set of transactions with $\langle T_i, \text{start} \rangle$ in log, but no $\langle T_i, \text{commit} \rangle$ (or $\langle T_i, \text{abort} \rangle$) record in log
- (2) For each $\langle T_i, X, v \rangle$ in log, in reverse order (latest \rightarrow earliest) do:
 - if $T_i \in S$ then {
 - write (X, v)
 - output (X)
- (3) For each $T_i \in S$ do
 - write $\langle T_i, \text{abort} \rangle$ to log

Question

- Can writes of $\langle T_i, \text{abort} \rangle$ records be done in any order (in Step 3)?
 - Example: T_1 and T_2 both write A
 - T_1 executed before T_2
 - T_1 and T_2 both rolled-back
 - $\langle T_1, \text{abort} \rangle$ written but NOT $\langle T_2, \text{abort} \rangle$?
 - $\langle T_2, \text{abort} \rangle$ written but NOT $\langle T_1, \text{abort} \rangle$?



What if failure during recovery?

No problem! ⇒ Undo idempotent

- An operation is called **idempotent** if the number of times it is applied do not effect the result
- For Undo:
 - Undo(log) = Undo(Undo(... (Undo(log)) ...))

Undo is idempotent

- We store the values of data items before the operation
- Undo can be executed repeatedly without changing effects
 - idempotent

Physical vs. Logical Logging

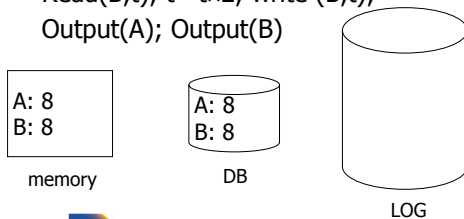
- How to represent values in log entries?
- Physical logging
 - Content of pages before and after
- Logical operations
 - Operation to execute for undo/redo
 - E.g., delete record x
- Hybrid (Physiological)
 - Delete record x from page y

To discuss:

- Redo logging
- Undo/redo logging, why both?
- Real world actions
- Checkpoints
- Media failures

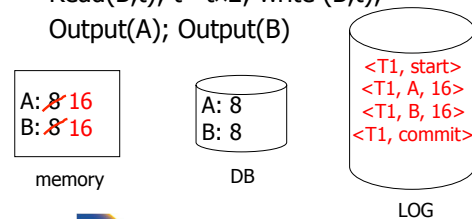
Redo logging (deferred modification)

T1: Read(A,t); t ← t×2; write (A,t);
Read(B,t); t ← t×2; write (B,t);
Output(A); Output(B)



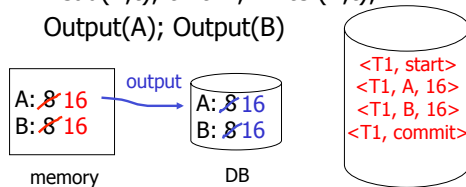
Redo logging (deferred modification)

T1: Read(A,t); t ← t×2; write (A,t);
Read(B,t); t ← t×2; write (B,t);
Output(A); Output(B)



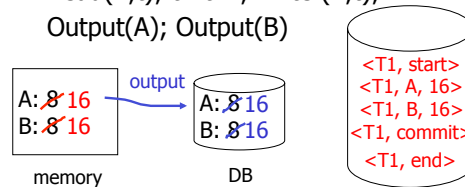
Redo logging (deferred modification)

T1: Read(A,t); t ← tx2; write (A,t);
 Read(B,t); t ← tx2; write (B,t);
 Output(A); Output(B)



Redo logging (deferred modification)

T1: Read(A,t); t ← tx2; write (A,t);
 Read(B,t); t ← tx2; write (B,t);
 Output(A); Output(B)



Redo logging rules

- (1) For every action, generate redo log record (containing new value)
- (2) Before X is modified on disk (DB), all log records for transaction that modified X (including commit) must be on disk
- (3) Flush log at commit
- (4) Write END record after DB updates flushed to disk

Recovery rules: Redo logging

- For every T_i with $\langle T_i, \text{commit} \rangle$ in log:
 - For all $\langle T_i, X, v \rangle$ in log:

$$\begin{cases} \text{Write}(X, v) \\ \text{Output}(X) \end{cases}$$

Recovery rules: Redo logging

- For every T_i with $\langle T_i, \text{commit} \rangle$ in log:
 - For all $\langle T_i, X, v \rangle$ in log:

$$\begin{cases} \text{Write}(X, v) \\ \text{Output}(X) \end{cases}$$

➡ IS THIS CORRECT??

Recovery rules: Redo logging

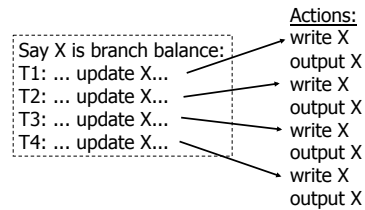
- (1) Let S = set of transactions with $\langle T_i, \text{commit} \rangle$ (and no $\langle T_i, \text{end} \rangle$) in log
- (2) For each $\langle T_i, X, v \rangle$ in log, in forward order (earliest → latest) do:
 - if $T_i \in S$ then $\begin{cases} \text{Write}(X, v) \\ \text{Output}(X) \end{cases}$
- (3) For each $T_i \in S$, write $\langle T_i, \text{end} \rangle$

Crash During Redo

- Since Redo log contains values after writes, repeated application of a log entry does not change result
--> idempotent

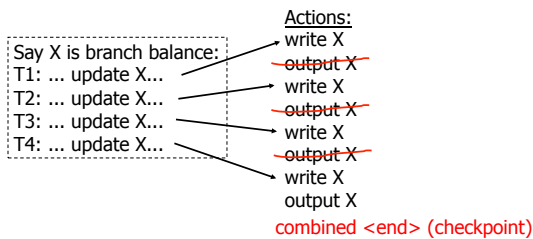
Combining <Ti, end> Records

- Want to delay DB flushes for hot objects



Combining <Ti, end> Records

- Want to delay DB flushes for hot objects



Solution: Checkpoint

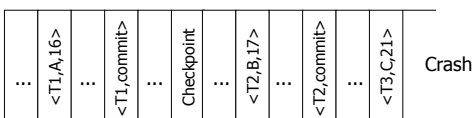
- no <ti, end> actions>
- simple checkpoint

Periodically:

- (1) Do not accept new transactions
- (2) Wait until all transactions finish
- (3) Flush all log records to disk (log)
- (4) Flush all buffers to disk (DB) (do not discard buffers)
- (5) Write "checkpoint" record on disk (log)
- (6) Resume transaction processing

Example: what to do at recovery?

Redo log (disk):



Advantage of Checkpoints

- Limits recovery to parts of the log after the checkpoint
 - Think about system that has been online for months
 - --> Analyzing the whole log is too expensive!
- Source of backups
 - If we backup checkpoints we can use them for media recovery!

Checkpoints Justification

- Checkpoint should be consistent DB state
 - No active transactions
 - Do not accept new transactions
 - Wait until all transactions finish
 - DB state reflected on disk
 - Flush log
 - Flush buffers

Key drawbacks:

- *Undo logging*:
 - cannot bring backup DB copies up to date
- *Redo logging*:
 - need to keep all modified blocks in memory until commit

Solution: undo/redo logging!

Update \Rightarrow $\langle T_i, Xid, \text{New } X \text{ val}, \text{Old } X \text{ val} \rangle$
page X

Rules

- Page X can be flushed before or after T_i commit
- Log record flushed before corresponding updated page (WAL)
- Flush at commit (log only)

Example: Undo/Redo logging
what to do at recovery?

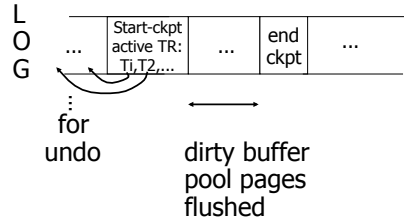
log (disk):

...	$\langle \text{checkpoint} \rangle$...	$\langle T_1, A, 10, 15 \rangle$...	$\langle T_1, B, 20, 23 \rangle$...	$\langle T_1, \text{commit} \rangle$...	$\langle T_2, C, 30, 38 \rangle$...	$\langle T_2, D, 40, 41 \rangle$	Crash
-----	-------------------------------------	-----	----------------------------------	-----	----------------------------------	-----	--------------------------------------	-----	----------------------------------	-----	----------------------------------	-------

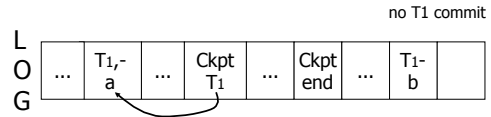
Checkpoint Cost

- Checkpoints are expensive
 - No new transactions can start
 - A lot of I/O
 - Flushing the log
 - Flushing dirty buffer pages

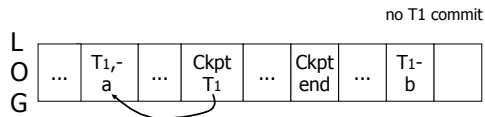
Non-quietse checkpoint



Examples what to do at recovery time?

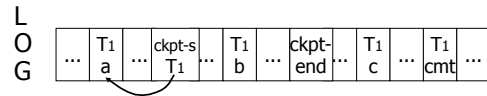


Examples what to do at recovery time?

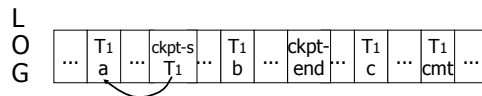


➡ Undo T1 (undo a,b)

Example

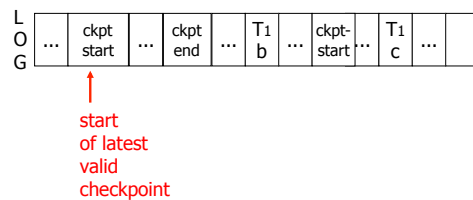


Example



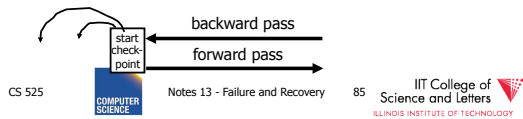
➡ Redo T1: (redo b,c)

Recover From Valid Checkpoint:



Recovery process:

- **Backwards pass** (end of log → latest valid checkpoint start)
 - construct set S of committed transactions
 - undo actions of transactions not in S
- **Undo pending transactions**
 - follow undo chains for transactions in (checkpoint active list) - S
- **Forward pass** (latest checkpoint start → end of log)
 - redo actions of S transactions



Real world actions

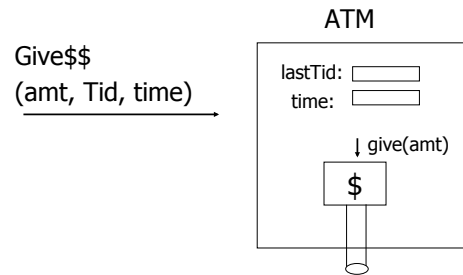
E.g., dispense cash at ATM

$$T_i = a_1 a_2 \dots a_j \dots a_n$$

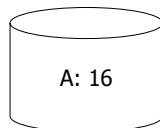
↓
\$

Solution

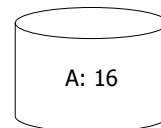
- (1) execute real-world actions after commit
- (2) try to make idempotent



Media failure (loss of non-volatile storage)



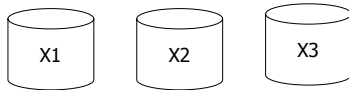
Media failure (loss of non-volatile storage)



Solution: Make copies of data!

Example 1 Triple modular redundancy

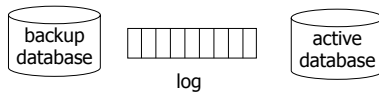
- Keep 3 copies on separate disks
- Output(X) --> three outputs
- Input(X) --> three inputs + vote



Example #2 Redundant writes, Single reads

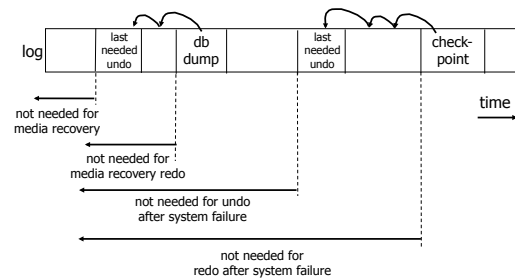
- Keep N copies on separate disks
 - Output(X) --> N outputs
 - Input(X) --> Input one copy
 - if ok, done
 - else try another one
- ⇒ Assumes bad data can be detected

Example #3: DB Dump + Log



- If active database is lost,
 - restore active database from backup
 - bring up-to-date using redo entries in log

When can log be discarded?



Practical Recovery with ARIES

- **ARIES**
 - Algorithms for **R**ecovery and **I**solation Exploiting **S**emantics
- Implemented in, e.g.,
 - DB2
 - MSSQL

Underlying Ideas

- Keep track of state of pages by relating them to entries in the log
- **WAL**
- Recovery in **three phases**
 - Analysis, Redo, Undo
- Log entries to track state of Undo for repeated failures
- **Redo**: page-oriented -> efficient
- **Undo**: logical -> permits higher level of concurrency

Log Entry Structure

- **LSN**

- **Log sequence number**
- Order of entries in the log
- Usually **log file id** and **offset** for direct access

- **LSN**

- **Entry type**

- Update, compensation, commit, ...

- **TID**

- Transaction identifier

- **PrevLSN**

- LSN of previous log record for same transaction

- **UndoNxtLSN**

- Next undo operation for CLR (later!)

- **Undo/Redo data**

- Data needed to undo/redo the update

Page Header Additions

- **PageLSN**

- **LSN** of the last update that modified the page
- Used to know which changes have been applied to a page

Forward Processing

- Normal operations when no ROLLBACK is required

- WAL: write redo/undo log record for each action of a transaction

- Buffer manager has to ensure that

- changes to pages are not persisted before the corresponding log record has been persisted
- Transactions are not considered committed before all their log records have been flushed

Dirty Page Table

- **PageLSN**

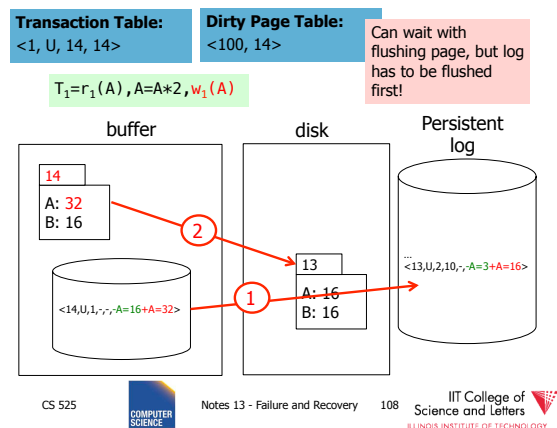
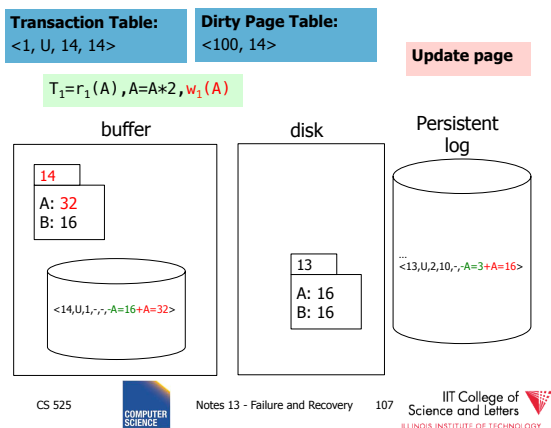
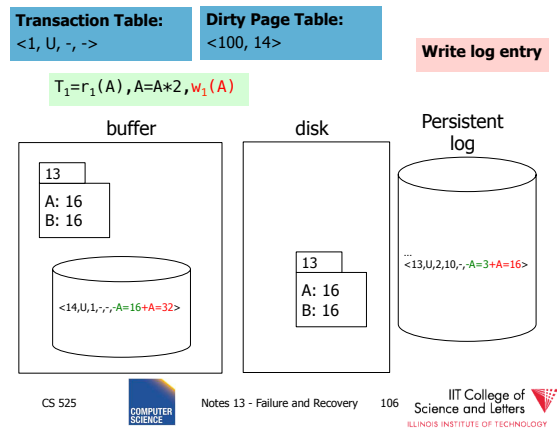
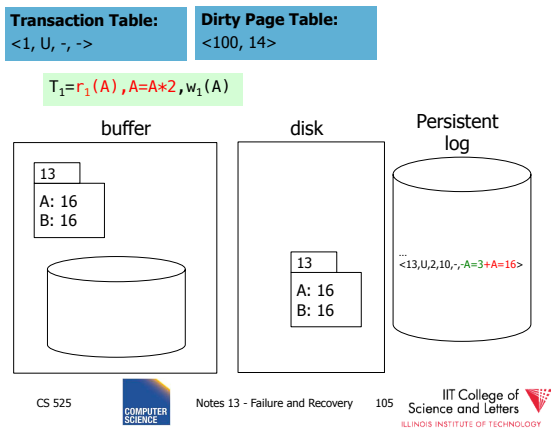
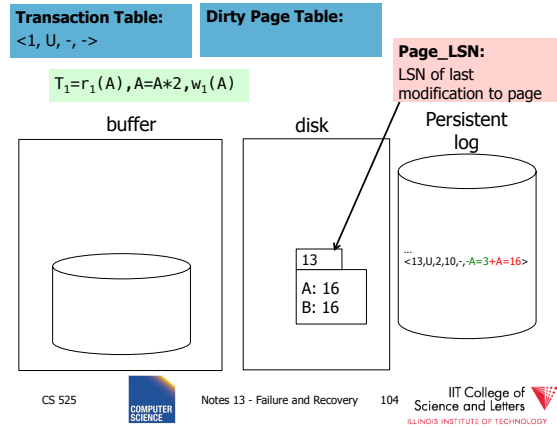
- Entries **<PageID,ReclSN>**
- Whenever a page is first fixed in the buffer pool with intention to modify
 - Insert **<PageId,ReclSN>** with **ReclSN** being the current end of the log
- Flushing a page removes it from the Dirty page table

Dirty Page Table

- Used for checkpointing
- Used for recovery to figure out what to redo

Transaction Table

- TransID
 - Identifier of the transaction
- State
 - Commit state
- LastLSN
 - LSN of the last update of the transaction
- UndoNxtLSN
 - If last log entry is a CLR then UndoNxtLSN from that record
 - Otherwise = LastLSN



Undo during forward processing

- Transaction was rolled back
 - User aborted, aborted because of error, ...
- Need to undo operations of transaction
- During Undo
 - Write log entries for every undo
 - Compensation Log Records (CLR)**
 - Used to avoid repeated undo when failures occur

Undo during forward processing

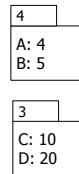
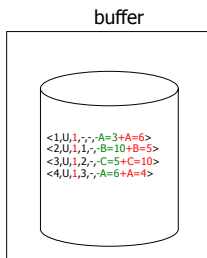
- Starting with the LastLSN of transaction from transaction table
 - Traverse log entries of transaction last to first using PrevLSN pointers
 - For each log entry use undo information to undo action
 - <LSN, Type, TID, PrevLSN, -, Undo/Redo data>
 - Before modifying data write an CLR that stores redo-information for the undo operation
 - UndoNxtLSN = PrevLSN of log entry we are undoing
 - Redo data = How to redo the undo

Transaction Table:

<1, U, 4, 4>

Undo T₁

T₁ = w₁(A), w₁(B), w₁(C), w₁(A), a₁

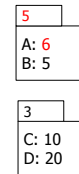
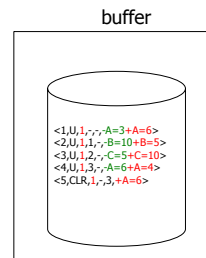


Transaction Table:

<1, U, 5, 3>

Undo T₁

T₁ = w₁(A), w₁(B), w₁(C), w₁(A), a₁

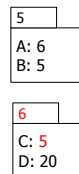
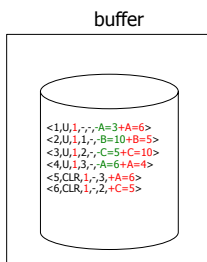


Transaction Table:

<1, U, 6, 2>

Undo T₁

T₁ = w₁(A), w₁(B), w₁(C), w₁(A), a₁

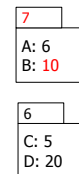
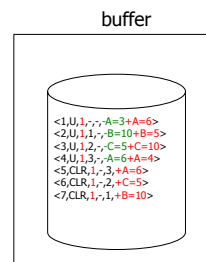


Transaction Table:

<1, U, 7, 1>

Undo T₁

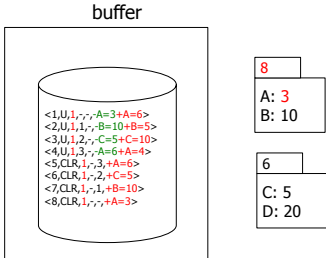
T₁ = w₁(A), w₁(B), w₁(C), w₁(A), a₁



Transaction Table:
 <1, U, 8, ->

Undo T_1

$T_1 = w_1(A), w_1(B), w_1(C), w_1(A), a_1$



CS 525



Notes 13 - Failure and Recovery

115



Fuzzy Checkpointing in ARIES

- Begin of checkpoint
 - Write **begin_cp** log entry
 - Write **end_cp** log entry with
 - Dirty page table
 - Transaction table
- **Master Record**
 - LSN of begin_cp log entry of last complete checkpoint

CS 525



Notes 13 - Failure and Recovery

116



Restart Recovery

1. Analysis Phase
2. Redo Phase
3. Undo Phase

CS 525



Notes 13 - Failure and Recovery

117



Analysis Phase

- 1) Determine LSN of last checkpoint using Master Record
- 2) Get Dirty Page Table and Transaction Table from checkpoint end record
- 3) **RedoLSN** = min(ReclSN) from Dirty Page Table or checkpoint LSN if no dirty page

CS 525



Notes 13 - Failure and Recovery

118



Analysis Phase

- 4) Scan log forward starting from RedoLSN
 - Update log entry from transaction
 - If necessary: Add Page to Dirty Page Table
 - Add Transaction to Transaction Table or update LastLSN
 - Transaction end entry
 - Remove transaction from Transaction Table

CS 525



Notes 13 - Failure and Recovery

119



Analysis Phase

- Result
 - Transaction Table
 - Transactions to be later undone
 - RedoLSN
 - Log entry to start Redo Phase
 - Dirty Page Table
 - Pages that may not have been written back to disk

CS 525



Notes 13 - Failure and Recovery

120



Redo Phase

- Start at RedoLSN scan log forward
- Unconditional Redo
 - Even redo actions of transactions that will be undone later
- Only redo once
 - Only redo operations that have not been reflected on disk (PageLSN)

Redo Phase

- For each update log entry
 - If affected page is not in Dirty Page Table or $ReLSN > LSN$
 - skip log entry
 - Fix page in buffer
 - If $PageLSN \geq LSN$ then operation already reflected on disk
 - Skip log entry
 - Otherwise apply update

Redo Phase

- Result
 - State of DB before Failure

Undo Phase

- Scan log backwards from end using Transaction Table
 - Repeatedly take log entry with max LSN from all the current actions to be undone for each transaction
 - Write CLR
 - Update Transaction Table

Undo Phase

- All unfinished transactions have been rolled back

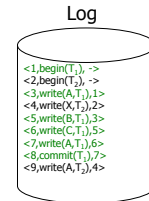
Idempotence?

- Redo
 - We are not logging during Redo so repeated Redo will result in the same state
- Undo
 - If we see CLR's we do not undo this action again

Avoiding Repeated Work

- Redo
 - If operation has been reflected on disk (PageLSN) we do not need to redo it again
- Undo
 - If we see CLR's we do not undo this action again

$T_1 = w_1(A), w_1(B), w_1(C), w_1(A), c_1$
 $T_2 = w_1(X), r(A), w(A)$



$T_1 = w_1(A), w_1(B), w_1(C), w_1(A), c_1$
 $T_2 = w_1(X), r(A), w(A)$

$T_1 = w_1(A), w_1(B), w_1(C), w_1(A), c_1$
 $T_2 = w_1(X), r(A), w(A)$

- Analysis Phase:**
- start at log entry 1
 - add T_1 to transaction table (rec. 1)
 - add T_2 to transaction table (rec. 2)
 - add A to dirty page table (ReclSN 3)
 - add X to dirty page table (ReclSN 4)
 - add B to dirty page table (ReclSN 5)
 - add C to dirty page table (ReclSN 6)
 - remove T1 from Transaction Table (rec. 8)



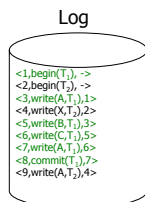
- Analysis Phase Result:**
- Transaction Table: $\langle T_2, 9 \rangle$
 - Dirty Page Table: $\langle A, 3 \rangle, \langle B, 5 \rangle, \langle C, 6 \rangle, \langle X, 4 \rangle$
 - RedoLSN = $\min(3, 5, 6, 4) = 3$



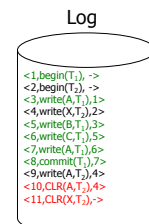
$T_1 = w_1(A), w_1(B), w_1(C), w_1(A), c_1$
 $T_2 = w_1(X), r(A), w(A)$

$T_1 = w_1(A), w_1(B), w_1(C), w_1(A), c_1$
 $T_2 = w_1(X), r(A), w(A)$

- Redo Phase (RedoLSN 3):**
- Read A if PageLSN < 3 apply write
 - Read X if PageLSN < 4 apply write
 - Read B if PageLSN < 5 apply write
 - Read C if PageLSN < 6 apply write
 - Read A if PageLSN < 7 apply write
 - Read A if PageLSN < 9 apply write





- Undo Phase (T_2):**
- Undo entry 9
 - write CLR with UndoNxtLSN = 4
 - modify page A
 - Undo entry 4
 - write CLR with UndoNxtLSN = 2
 - modify page X
 - Done





ARIES take away messages

- Provide good performance by
 - Not requiring complete checkpoints
 - Linking of log records
 - Not restricting buffer operations (no-force/steal is ok)
- Logical Undo and Physical (Physiological) Redo
- Idempotent Redo and Undo
 - Avoid undoing the same operation twice

CS 525  Notes 13 - Failure and Recovery 133  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY



Media Recovery

- What if disks where log or DB is stored fails
 - ->keep backups of log + DB state

CS 525  Notes 13 - Failure and Recovery 134  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY



Log Backup

- Split log into several files
- Is append only, backup of old files cannot interfere with current log operations

CS 525  Notes 13 - Failure and Recovery 135  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY



Backup DB state

- Copy current DB state directly from disk
- May be inconsistent
- ->Use log to know which pages are up-to-date and redo operations not yet reflected

CS 525  Notes 13 - Failure and Recovery 136  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Summary

- Consistency of data
- One source of problems: failures
 - Logging
 - Redundancy
- Another source of problems:
Data Sharing..... next

CS 525  Notes 13 - Failure and Recovery 137  IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Chapter 18 [18] Concurrency Control

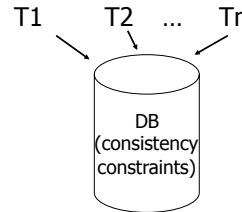
CS 525: Advanced Database Organization

14: Concurrency Control



Boris Glavic

Slides: adapted from a [course](#) taught by [Hector Garcia-Molina](#), Stanford InfoLab



CS 525



Notes 14 - Concurrency Control

1 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

CS 525



Notes 14 - Concurrency Control

2 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

Example:

T1: Read(A)	T2: Read(A)
A ← A+100	A ← A×2
Write(A)	Write(A)
Read(B)	Read(B)
B ← B+100	B ← B×2
Write(B)	Write(B)

Constraint: A=B

Schedule A

T1	T2
Read(A); A ← A+100	
Write(A);	
Read(B); B ← B+100;	
Write(B);	
	Read(A); A ← A×2;
	Write(A);
	Read(B); B ← B×2;
	Write(B);

CS 525



Notes 14 - Concurrency Control

3 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

CS 525



Notes 14 - Concurrency Control

4 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

Schedule A

T1	T2	A	B
Read(A); A ← A+100		25	25
Write(A);			
Read(B); B ← B+100;		125	
Write(B);			
	Read(A); A ← A×2;		
	Write(A);	250	
	Read(B); B ← B×2;		
	Write(B);		250
		250	250

CS 525



Notes 14 - Concurrency Control

5 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

Schedule B

T1	T2
	Read(A); A ← A×2;
	Write(A);
	Read(B); B ← B×2;
	Write(B);
Read(A); A ← A+100	
Write(A);	
Read(B); B ← B+100;	
Write(B);	

CS 525



Notes 14 - Concurrency Control

6 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

Schedule B

T1	T2	A	B
		25	25
	Read(A); A ← A×2;		
	Write(A);	50	
	Read(B); B ← B×2;		
	Write(B);		50
Read(A); A ← A+100			
Write(A);		150	
Read(B); B ← B+100;			
Write(B);			150
		150	150

CS 525



Notes 14 - Concurrency Control

7

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Schedule C

T1	T2
Read(A); A ← A+100	
Write(A);	
	Read(A); A ← A×2;
	Write(A);
Read(B); B ← B+100;	
Write(B);	
	Read(B); B ← B×2;
	Write(B);

CS 525



Notes 14 - Concurrency Control

8

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Schedule C

T1	T2	A	B
		25	25
Read(A); A ← A+100			
Write(A);		125	
	Read(A); A ← A×2;		
	Write(A);	250	
Read(B); B ← B+100;			
Write(B);			125
	Read(B); B ← B×2;		
	Write(B);		250
		250	250

CS 525



Notes 14 - Concurrency Control

9

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Schedule D

T1	T2
Read(A); A ← A+100	
Write(A);	
	Read(A); A ← A×2;
	Write(A);
	Read(B); B ← B×2;
	Write(B);
Read(B); B ← B+100;	
Write(B);	

CS 525



Notes 14 - Concurrency Control

10

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Schedule D

T1	T2	A	B
		25	25
Read(A); A ← A+100			
Write(A);		125	
	Read(A); A ← A×2;		
	Write(A);	250	
	Read(B); B ← B×2;		
	Write(B);		50
Read(B); B ← B+100;			
Write(B);			150
		250	150

CS 525



Notes 14 - Concurrency Control

11

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Schedule E

Same as Schedule D but with new T2'

T1	T2'
Read(A); A ← A+100	
Write(A);	
	Read(A); A ← A×1;
	Write(A);
	Read(B); B ← B×1;
	Write(B);
Read(B); B ← B+100;	
Write(B);	

CS 525



Notes 14 - Concurrency Control

12

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Schedule E

Same as Schedule D but with new T2'

T1	T2'	A	B
Read(A); A ← A+100		25	25
Write(A);		125	
	Read(A); A ← A×1;	125	
	Write(A);		25
	Read(B); B ← B×1;		125
	Write(B);		125
Read(B); B ← B+100;		125	
Write(B);		125	

CS 525



Notes 14 - Concurrency Control

13

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Serial Schedules

- As long as we do not execute transactions in parallel and each transaction does not violate the constraints we are good
 - All schedules with no interleaving of transaction operations are called **serial** schedules

CS 525



Notes 14 - Concurrency Control

14

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Definition: Serial Schedule

- No transactions are interleaved
 - There exists no two operations from transactions T_i and T_j so that both operations are executed before either transaction commits

$$T_1 = r_1(A), w_1(A), r_1(B), w_1(B), c_1$$

$$T_2 = r_2(A), w_2(A), r_2(B), w_2(B), c_2$$

Serial Schedule

$$S_1 = r_2(A), w_2(A), r_2(B), w_2(B), c_2, r_1(A), w_1(A), r_1(B), w_1(B), c_1$$

Nonserial Schedule

$$S_2 = r_2(A), w_2(A), r_1(A), w_1(A), r_2(B), w_2(B), c_2, r_1(B), w_1(B), c_1$$

CS 525



Notes 14 - Concurrency Control

15

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

CS 525



Notes 12 - Transaction Management

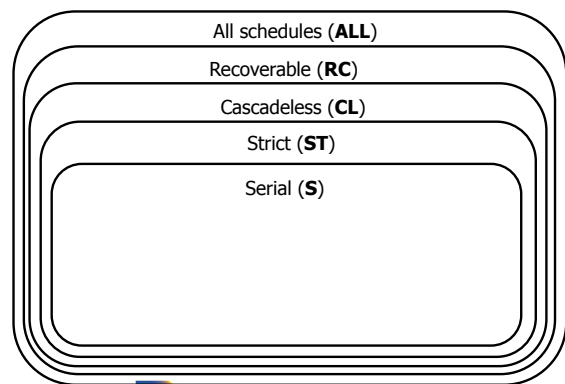
16

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Compare Classes

$$S \subset ST \subset CL \subset RC \subset ALL$$

- Abbreviations
 - S = Serial
 - ST = Strict
 - CL = Cascadeless
 - RC = Recoverable
 - ALL = all possible schedules



CS 525



Notes 13 - Failure and Recovery

17

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

CS 525



Notes 13 - Failure and Recovery

18

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Why not serial schedules?

- No concurrency! ☹️

- Want schedules that are “good”, regardless of
 - initial state and
 - transaction semantics
- Only look at order of read and writes

Example:

$Sc = r_1(A)w_1(A)r_2(A)w_2(A)r_1(B)w_1(B)r_2(B)w_2(B)$

Outline

- Since serial schedules have good properties we would like our schedules to behave like (be **equivalent** to) serial schedules
 1. Need to define equivalence based solely on order of operations
 2. Need to define class of schedules which is equivalent to serial schedule
 3. Need to design scheduler that guarantees that we only get these good schedules

Example:

$Sc = r_1(A)w_1(A)r_2(A)w_2(A)r_1(B)w_1(B)r_2(B)w_2(B)$

$Sc' = r_1(A)w_1(A) \underbrace{r_1(B)w_1(B)r_2(A)w_2(A)}_{T_1} \underbrace{r_2(B)w_2(B)}_{T_2}$

However, for S_d :

$S_d = r_1(A)w_1(A)r_2(A)w_2(A) \underbrace{r_2(B)w_2(B)r_1(B)w_1(B)}_{T_1}$

- as a matter of fact, T_2 must precede T_1 in any equivalent schedule, i.e., $T_2 \rightarrow T_1$

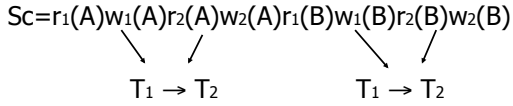
- $T_2 \rightarrow T_1$
- Also, $T_1 \rightarrow T_2$

$T_1 \rightarrow T_2$ $\Rightarrow S_d$ cannot be rearranged into a serial schedule

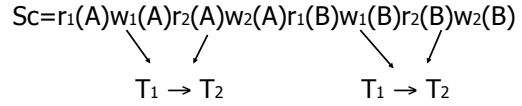
$\Rightarrow S_d$ is not “equivalent” to any serial schedule

$\Rightarrow S_d$ is “bad”

Returning to Sc



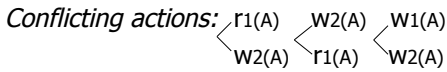
Returning to Sc



no cycles \Rightarrow Sc is "equivalent" to a serial schedule (in this case T_1, T_2)

Concepts

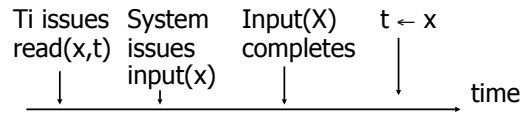
Transaction: sequence of $r_i(x), w_i(x)$ actions



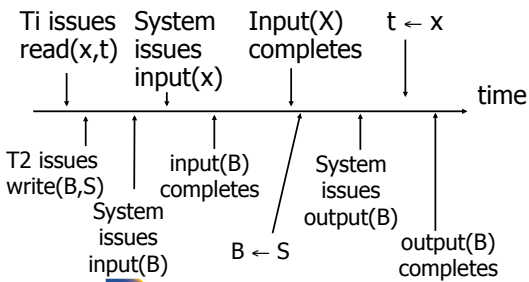
Schedule: represents chronological order in which actions are executed

Serial schedule: no interleaving of actions or transactions

What about concurrent actions?



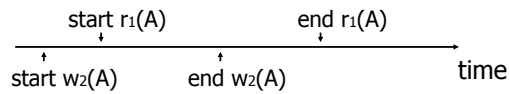
What about concurrent actions?



So net effect is either

- $S = \dots r_1(x) \dots w_2(b) \dots$ or
- $S = \dots w_2(B) \dots r_1(x) \dots$

What about conflicting, concurrent actions on same object?



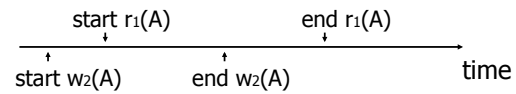
CS 525



Notes 14 - Concurrency Control

31

What about conflicting, concurrent actions on same object?



- Assume equivalent to either $r_1(A) w_2(A)$ or $w_2(A) r_1(A)$
- \Rightarrow low level synchronization mechanism
- Assumption called “atomic actions”

CS 525



Notes 14 - Concurrency Control

32

Outline

- Since serial schedules have good properties we would like our schedules to behave like (be **equivalent** to) serial schedules
 1. Need to define equivalence based solely on order of operations
 2. Need to define class of schedules which is equivalent to serial schedule
 3. Need to design scheduler that guarantees that we only get these good schedules

CS 525



Notes 14 - Concurrency Control

33

Conflict Equivalence

- Define equivalence based on the order of conflicting actions

CS 525



Notes 14 - Concurrency Control

34

Definition

S_1, S_2 are conflict equivalent schedules if S_1 can be transformed into S_2 by a series of swaps on non-conflicting actions.

Alternatively:

If the order of conflicting actions in S_1 and S_2 is the same

CS 525



Notes 14 - Concurrency Control

35

Outline

- Since serial schedules have good properties we would like our schedules to behave like (be **equivalent** to) serial schedules
 1. Need to define equivalence based solely on order of operations
 2. Need to define class of schedules which is equivalent to serial schedule
 3. Need to design scheduler that guarantees that we only get these good schedules

CS 525



Notes 14 - Concurrency Control

36

Definition

A schedule is conflict serializable (CSR) if it is conflict equivalent to some serial schedule.

How to check?

- Compare orders of all conflicting operations
- Can be simplified because there is some redundant information here, e.g.,

$$S_1 = w_2(A), w_2(B), r_1(A), w_1(B)$$

- $W_2(A)$ conflicts with $R_1(A)$
- $W_2(B)$ conflicts with $W_1(B)$
- Both imply that T_2 has to be executed before T_1 in any equivalent serial schedule

Conflict graph $P(S)$ (S is schedule)

Nodes: transactions in S

Arcs: $T_i \rightarrow T_j$ whenever

- $p_i(A), q_j(A)$ are actions in S
- $p_i(A) <_S q_j(A)$
- at least one of p_i, q_j is a write

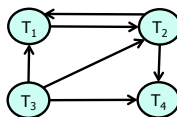
Exercise:

- What is $P(S)$ for $S = w_3(A) w_2(C) r_1(A) w_1(B) r_1(C) w_2(A) r_4(A) w_4(D)$

- Is S serializable?

Exercise:

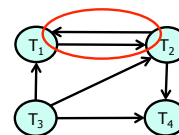
- What is $P(S)$ for $S = w_3(A) w_2(C) r_1(A) w_1(B) r_1(C) w_2(A) r_4(A) w_4(D)$



- Is S serializable?

Exercise:

- What is $P(S)$ for $S = w_3(A) w_2(C) r_1(A) w_1(B) r_1(C) w_2(A) r_4(A) w_4(D)$



- Is S serializable?

Another Exercise:

- What is P(S) for
 $S = w_1(A) r_2(A) r_3(A) w_4(A)$?

CS 525



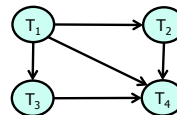
Notes 14 - Concurrency Control

43



Another Exercise:

- What is P(S) for
 $S = w_1(A) r_2(A) r_3(A) w_4(A)$?



CS 525



Notes 14 - Concurrency Control

44



Lemma

S_1, S_2 conflict equivalent $\Rightarrow P(S_1)=P(S_2)$

CS 525



Notes 14 - Concurrency Control

45



Lemma

S_1, S_2 conflict equivalent $\Rightarrow P(S_1)=P(S_2)$

Proof: (a \rightarrow b same as \neg b \rightarrow \neg a)

Assume $P(S_1) \neq P(S_2)$

$\Rightarrow \exists T_i: T_i \rightarrow T_j$ in S_1 and not in S_2

$\Rightarrow S_1 = \dots p_i(A) \dots q_j(A) \dots$
 $S_2 = \dots q_j(A) \dots p_i(A) \dots$ } p_i, q_j
 } conflict

$\Rightarrow S_1, S_2$ not conflict equivalent

CS 525



Notes 14 - Concurrency Control

46



Note: $P(S_1)=P(S_2) \not\Rightarrow S_1, S_2$ conflict equivalent

Note: $P(S_1)=P(S_2) \not\Rightarrow S_1, S_2$ conflict equivalent

Counter example:

$S_1 = w_1(A) r_2(A) w_2(B) r_1(B)$

$S_2 = r_2(A) w_1(A) r_1(B) w_2(B)$

CS 525



Notes 14 - Concurrency Control

47



CS 525



Notes 14 - Concurrency Control

48



Theorem

$P(S_1)$ acyclic $\iff S_1$ conflict serializable

(\Leftarrow) Assume S_1 is conflict serializable

$\Rightarrow \exists S_s: S_s, S_1$ conflict equivalent

$\Rightarrow P(S_s) = P(S_1)$

$\Rightarrow P(S_1)$ acyclic since $P(S_s)$ is acyclic

CS 525



Notes 14 - Concurrency Control

49

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Theorem

$P(S_1)$ acyclic $\iff S_1$ conflict serializable

(\Rightarrow) Assume $P(S_1)$ is acyclic

Transform S_1 as follows:

- (1) Take T_1 to be transaction with no incident arcs
- (2) Move all T_1 actions to the front

$S_1 = \dots q_j(A) \dots p_1(A) \dots$



- (3) we now have $S_1 = \langle T_1 \text{ actions} \rangle \dots \text{rest} \dots$
- (4) repeat above steps to serialize rest!

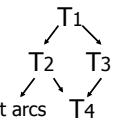
CS 525



Notes 14 - Concurrency Control

50

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY



What's the damage?

- Classification of "bad" things that can happen in "bad" schedules
 - Dirty reads
 - Non-repeatable reads
 - Phantom reads (later)

CS 525



Notes 14 - Concurrency Control

51

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Dirty Read

- A transaction T_1 read a value that has been updated by an uncommitted transaction T_2
- If T_2 aborts then the value read by T_1 is invalid

CS 525

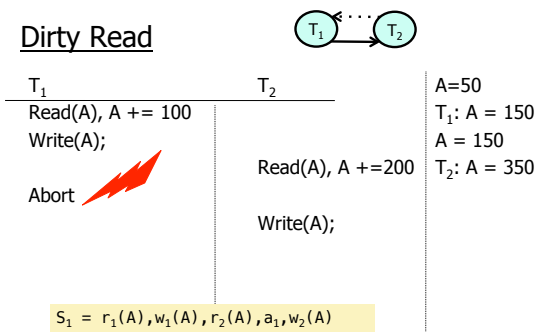


Notes 14 - Concurrency Control

52

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Dirty Read



CS 525



Notes 14 - Concurrency Control

53

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Non-repeatable Read

- A transaction T_1 reads items; some before and some after an update of these item by a transaction T_2
- Problem
 - Repeated reads of the same item see different values
 - Some values are modified and some are not

CS 525

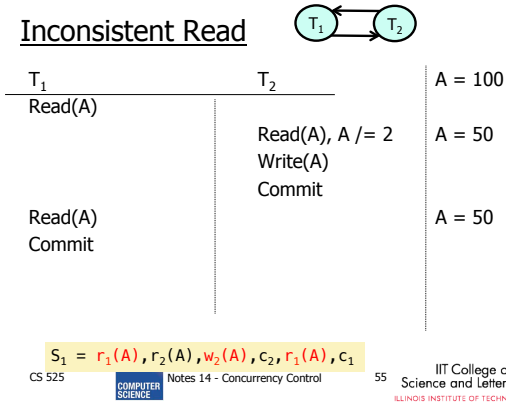


Notes 14 - Concurrency Control

54

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Inconsistent Read



How to enforce serializable schedules?

Option 1: run system, recording P(S);
at end of day, check for P(S)
cycles and declare if execution
was good

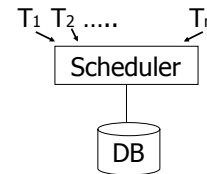
How to enforce serializable schedules?

Option 1: run system, recording P(S);
at end of day, check for P(S)
cycles and declare if execution
was good

This is called **optimistic concurrency control**

How to enforce serializable schedules?

Option 2: prevent P(S) cycles from occurring



How to enforce serializable schedules?

Option 2: prevent P(S) cycles from occurring

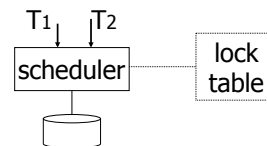
This is called **pessimistic concurrency control**

A locking protocol

Two new actions:

lock (exclusive): $li(A)$

unlock: $ui(A)$



Rule #1: Well-formed transactions

Ti: ... li(A) ... pi(A) ... ui(A) ...

- 1) Transaction has to lock A before it can access A
- 2) Transaction has to unlock A eventually
- 3) Transaction cannot access A after unlock

Rule #2 Legal scheduler

S = li(A) ui(A)
 $\xrightarrow{\hspace{2cm}}$
 no lj(A)

- 4) Only one transaction can hold a lock on A at the same time

Exercise:

- What schedules are legal? What transactions are well-formed?

$S_1 = l_1(A)l_1(B)r_1(A)w_1(B)l_2(B)u_1(A)u_1(B)$
 $r_2(B)w_2(B)u_2(B)l_3(B)r_3(B)u_3(B)$
 $S_2 = l_1(A)r_1(A)w_1(B)u_1(A)u_1(B)$
 $l_2(B)r_2(B)w_2(B)l_3(B)r_3(B)u_3(B)$
 $S_3 = l_1(A)r_1(A)u_1(A)l_1(B)w_1(B)u_1(B)$
 $l_2(B)r_2(B)w_2(B)u_2(B)l_3(B)r_3(B)u_3(B)$

Exercise:

- What schedules are legal? What transactions are well-formed?

$S_1 = l_1(A)l_1(B)r_1(A)w_1(B)l_2(B)u_1(A)u_1(B)$
 $r_2(B)w_2(B)u_2(B)l_3(B)r_3(B)u_3(B)$
 $S_2 = l_1(A)r_1(A)w_1(B)u_1(A)u_1(B)$
 $l_2(B)r_2(B)w_2(B)l_3(B)r_3(B)u_3(B)$
 $S_3 = l_1(A)r_1(A)u_1(A)l_1(B)w_1(B)u_1(B)$
 $l_2(B)r_2(B)w_2(B)u_2(B)l_3(B)r_3(B)u_3(B)$

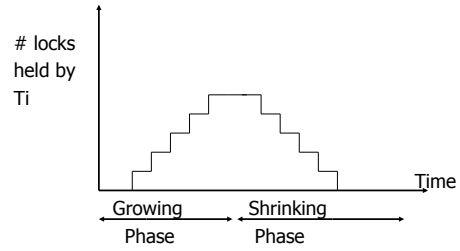
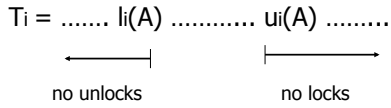
Schedule F

T1	T2
$l_1(A); \text{Read}(A)$	
$A \leftarrow A+100; \text{Write}(A); u_1(A)$	
	$l_2(A); \text{Read}(A)$
	$A \leftarrow Ax2; \text{Write}(A); u_2(A)$
	$l_2(B); \text{Read}(B)$
	$B \leftarrow Bx2; \text{Write}(B); u_2(B)$
$l_1(B); \text{Read}(B)$	
$B \leftarrow B+100; \text{Write}(B); u_1(B)$	

Schedule F

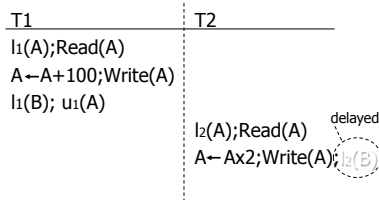
T1	T2	A	B
$l_1(A); \text{Read}(A)$		25	25
$A \leftarrow A+100; \text{Write}(A); u_1(A)$		125	
	$l_2(A); \text{Read}(A)$		
	$A \leftarrow Ax2; \text{Write}(A); u_2(A)$	250	
	$l_2(B); \text{Read}(B)$		
	$B \leftarrow Bx2; \text{Write}(B); u_2(B)$		50
$l_1(B); \text{Read}(B)$			150
$B \leftarrow B+100; \text{Write}(B); u_1(B)$		250	150

Rule #3 Two phase locking (2PL)
for transactions

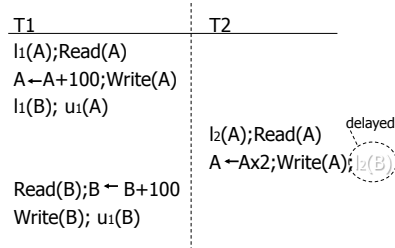


5) A transaction does not require new locks after its first unlock operation

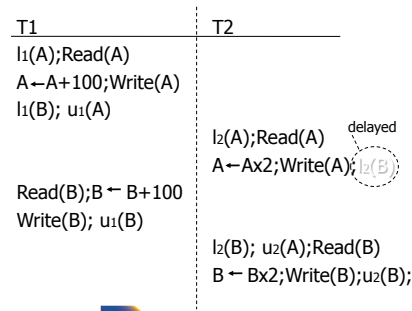
Schedule G



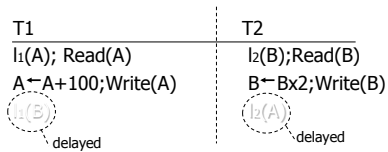
Schedule G



Schedule G



Schedule H (T2 reversed)



Deadlock

- Two or more transactions are waiting for each other to release a lock
- In the example
 - T_1 is waiting for T_2 and is making no progress
 - T_2 is waiting for T_1 and is making no progress
 - -> if we do not do anything they would wait forever

- Assume deadlocked transactions are rolled back
 - They have no effect
 - They do not appear in schedule
 - **Come back to that later**

E.g., Schedule H = _____
This space intentionally left blank!

Next step:

Show that rules #1,2,3 \Rightarrow conflict-serializable schedules

Conflict rules for $l_i(A), u_i(A)$:

- $l_i(A), l_j(A)$ conflict
- $l_i(A), u_j(A)$ conflict

Note: no conflict $\langle u_i(A), u_j(A) \rangle, \langle l_i(A), r_j(A) \rangle, \dots$

Theorem Rules #1,2,3 \Rightarrow conflict serializable schedule (2PL)

Theorem Rules #1,2,3 \Rightarrow conflict serializable schedule (2PL)

To help in proof:

Definition $\text{Shrink}(T_i) = \text{SH}(T_i) =$
first unlock action of T_i

Lemma

$T_i \rightarrow T_j \text{ in } S \Rightarrow SH(T_i) <_S SH(T_j)$

Lemma

$T_i \rightarrow T_j \text{ in } S \Rightarrow SH(T_i) <_S SH(T_j)$

Proof of lemma:

$T_i \rightarrow T_j$ means that

$S = \dots p_i(A) \dots q_j(A) \dots;$ p, q conflict

By rules 1,2:

$S = \dots p_i(A) \dots u_i(A) \dots l_j(A) \dots q_j(A) \dots$

Lemma

$T_i \rightarrow T_j \text{ in } S \Rightarrow SH(T_i) <_S SH(T_j)$

Proof of lemma:

$T_i \rightarrow T_j$ means that

$S = \dots p_i(A) \dots q_j(A) \dots;$ p, q conflict

By rules 1,2:

$S = \dots p_i(A) \dots u_i(A) \dots l_j(A) \dots q_j(A) \dots$

By rule 3: $SH(T_i) \quad SH(T_j)$

So, $SH(T_i) <_S SH(T_j)$

Theorem Rules #1,2,3 \Rightarrow conflict (2PL) serializable schedule

Proof:

(1) Assume P(S) has cycle

$T_1 \rightarrow T_2 \rightarrow \dots \rightarrow T_n \rightarrow T_1$

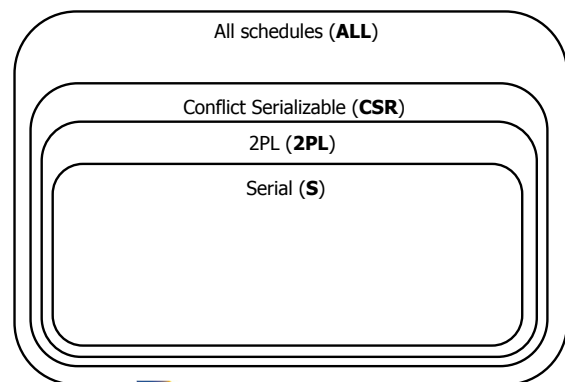
(2) By lemma: $SH(T_1) < SH(T_2) < \dots < SH(T_1)$

(3) Impossible, so P(S) acyclic

(4) $\Rightarrow S$ is conflict serializable

2PL subset of Serializable

$S \subset 2PL \subset CSRC \subset ALL$



S1: w1(x) w3(x) w2(y) w1(y)

- S1 cannot be achieved via 2PL:
The lock by T1 for y must occur after w2(y), so the unlock by T1 for x must occur after this point (and before w1(x)). Thus, w3(x) cannot occur under 2PL where shown in S1 because T1 holds the x lock at that point.
- However, S1 is serializable (equivalent to T2, T1, T3).

If you need a bit more practice:

Are our schedules S_C and S_D 2PL schedules?

S_C : w1(A) w2(A) w1(B) w2(B)

S_D : w1(A) w2(A) w2(B) w1(B)

- Beyond this simple **2PL** protocol, it is all a matter of improving performance and allowing more concurrency....
 - Shared locks
 - Multiple granularity
 - Avoid Deadlocks
 - Inserts, deletes and phantoms
 - Other types of C.C. mechanisms
 - Multiversioning concurrency control

Shared locks

So far:

$S = \dots l_1(A) r_1(A) u_1(A) \dots l_2(A) r_2(A) u_2(A) \dots$

Do not conflict

Shared locks

So far:

$S = \dots l_1(A) r_1(A) u_1(A) \dots l_2(A) r_2(A) u_2(A) \dots$

Do not conflict

Instead:

$S = \dots ls_1(A) r_1(A) ls_2(A) r_2(A) \dots us_1(A) us_2(A)$

Lock actions

$l-t(A)$: lock A in t mode (t is S or X)

$u-t(A)$: unlock t mode (t is S or X)

Shorthand:

$u_i(A)$: unlock whatever modes

T_i has locked A

Rule #1 Well formed transactions

$T_i = \dots I-S_i(A) \dots r_1(A) \dots u_1(A) \dots$
 $T_i = \dots I-X_i(A) \dots w_1(A) \dots u_1(A) \dots$

- What about transactions that read and write same object?

Option 1: Request exclusive lock

$T_i = \dots I-X_i(A) \dots r_1(A) \dots w_1(A) \dots u_1(A) \dots$

CS 525



Notes 14 - Concurrency Control

91



CS 525



Notes 14 - Concurrency Control

92



- What about transactions that read and write same object?

Option 2: Upgrade

(E.g., need to read, but don't know if will write...)

$T_i = \dots I-S_i(A) \dots r_1(A) \dots I-X_i(A) \dots w_1(A) \dots u_1(A) \dots$

Think of
 - Get 2nd lock on A, or
 - Drop S, get X lock

Rule #2 Legal scheduler

$S = \dots I-S_i(A) \dots \dots u_i(A) \dots$
 no $I-X_j(A)$

$S = \dots I-X_i(A) \dots \dots u_i(A) \dots$
 no $I-X_j(A)$
 no $I-S_j(A)$

CS 525



Notes 14 - Concurrency Control

93



CS 525



Notes 14 - Concurrency Control

94



A way to summarize Rule #2

Compatibility matrix

Comp		S	X
S		true	false
X		false	false

Rule # 3 2PL transactions

No change except for upgrades:

- (I) If upgrade gets more locks (e.g., $S \rightarrow \{S, X\}$) then no change!
- (II) If upgrade releases read (shared) lock (e.g., $S \rightarrow X$) - can be allowed in growing phase

CS 525



Notes 14 - Concurrency Control

95



CS 525



Notes 14 - Concurrency Control

96



Theorem Rules 1,2,3 \Rightarrow Conf.serializable
for S/X locks schedules

Proof: similar to X locks case

Detail:

$l-t_i(A), l-r_j(A)$ do not conflict if $\text{comp}(t,r)$

$l-t_i(A), u-r_j(A)$ do not conflict if $\text{comp}(t,r)$

Lock types beyond S/X

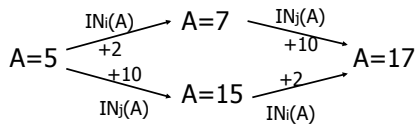
Examples:

(1) increment lock

(2) update lock

Example (1): increment lock

- Atomic increment action: $IN_i(A)$
 $\{ \text{Read}(A); A \leftarrow A+k; \text{Write}(A) \}$
- $IN_i(A), IN_j(A)$ do not conflict!



Comp

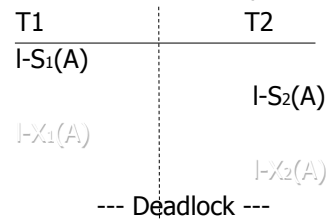
	S	X	I
S			
X			
I			

Comp

	S	X	I
S	T	F	F
X	F	F	F
I	F	F	T

Update locks

A common deadlock problem with upgrades:



Solution

If T_i wants to read A and knows it may later want to write A, it requests update lock (not shared)

		New request			
Comp	Lock already held in		S	X	U
		S			
		X			
		U			

		New request			
Comp	Lock already held in		S	X	U
		S	T	F	T
		X	F	F	F
		U	TorF	F	F

-> symmetric table?

Note: object A may be locked in different modes at the same time...

$$S_1 = \dots I-S_1(A) \dots I-S_2(A) \dots I-U_3(A) \dots \left\{ \begin{array}{l} I-S_4(A) \dots? \\ I-U_4(A) \dots? \end{array} \right.$$

Note: object A may be locked in different modes at the same time...

$$S_1 = \dots I-S_1(A) \dots I-S_2(A) \dots I-U_3(A) \dots \left\{ \begin{array}{l} I-S_4(A) \dots? \\ I-U_4(A) \dots? \end{array} \right.$$

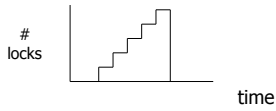
- To grant a lock in mode t , mode t must be compatible with all currently held locks on object

How does locking work in practice?

- Every system is different (E.g., may not even provide CONFLICT-SERIALIZABLE schedules)
- But here is one (simplified) way ...

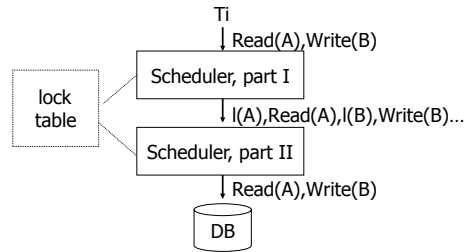
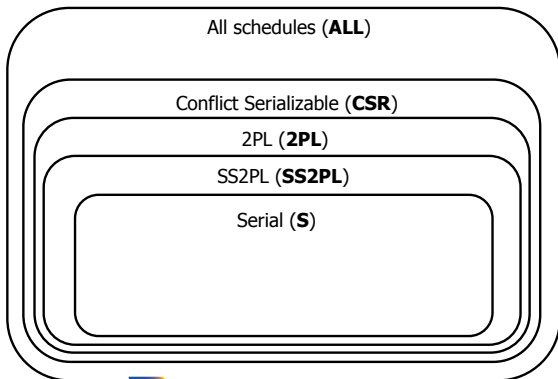
Sample Locking System:

- (1) Don't trust transactions to request/release locks
- (2) Hold all locks until transaction commits

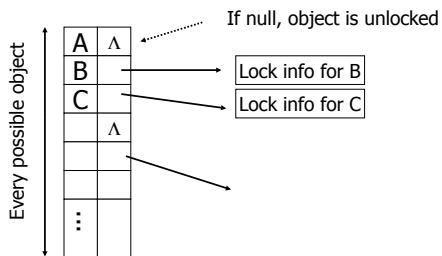


Strict Strong 2PL (SS2PL)

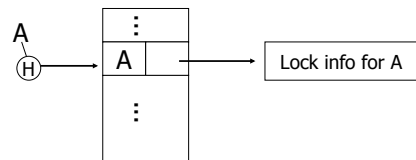
- 2PL + (2) from the last slide
- All locks are held until transaction end
- Compare with schedule class **strict (ST)** we defined for recovery
 - A transaction never reads or writes items written by an uncommitted transactions
- **SS2PL = (ST ∩ 2PL)**



Lock table Conceptually

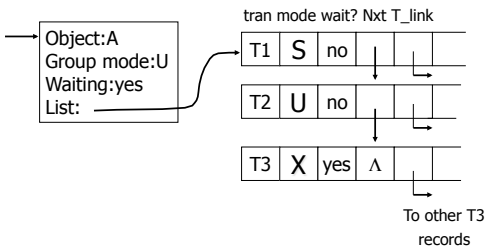


But use hash table:



If object not found in hash table, it is unlocked

Lock info for A - example



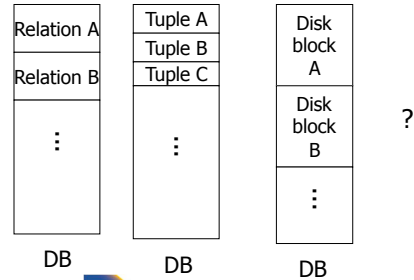
CS 525



Notes 14 - Concurrency Control

115 IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

What are the objects we lock?



CS 525



Notes 14 - Concurrency Control

116 IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

- Locking works in any case, but should we choose small or large objects?

- Locking works in any case, but should we choose small or large objects?

- If we lock large objects (e.g., Relations)
 - Need few locks
 - Low concurrency
- If we lock small objects (e.g., tuples, fields)
 - Need more locks
 - More concurrency

CS 525



Notes 14 - Concurrency Control

117 IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

CS 525

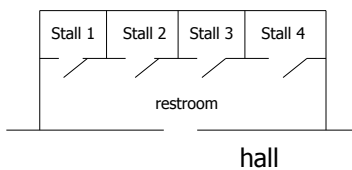


Notes 14 - Concurrency Control

118 IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

We can have it both ways!!

Ask any janitor to give you the solution...



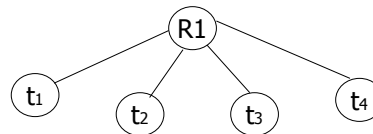
CS 525



Notes 14 - Concurrency Control

119 IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Example



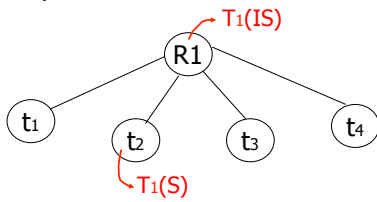
CS 525



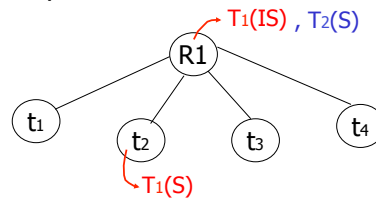
Notes 14 - Concurrency Control

120 IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

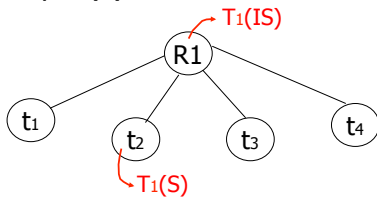
Example



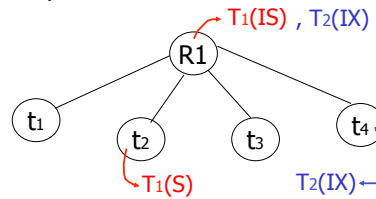
Example



Example (b)



Example



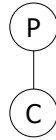
Multiple granularity

Comp	Requestor				
	IS	IX	S	SIX	X
Holder	IS				
	IX				
	S				
	SIX				
	X				

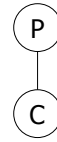
Multiple granularity

Comp	Requestor				
	IS	IX	S	SIX	X
Holder	IS	T	T	T	F
	IX	T	T	F	F
	S	T	F	T	F
	SIX	T	F	F	F
	X	F	F	F	F

Parent locked in	Child can be locked in
IS	
IX	
S	
SIX	
X	



Parent locked in	Child can be locked by same transaction in
IS	IS, S
IX	IS, S, IX, X, SIX
S	none
SIX	X, IX, [SIX]
X	none



not necessary

CS 525



Notes 14 - Concurrency Control

127

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

CS 525



Notes 14 - Concurrency Control

128

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Rules

- (1) Follow multiple granularity comp function
- (2) Lock root of tree first, any mode
- (3) Node Q can be locked by Ti in S or IS only if parent(Q) locked by Ti in IX or IS
- (4) Node Q can be locked by Ti in X,SIX,IX only if parent(Q) locked by Ti in IX,SIX
- (5) Ti is two-phase
- (6) Ti can unlock node Q only if none of Q's children are locked by Ti

CS 525



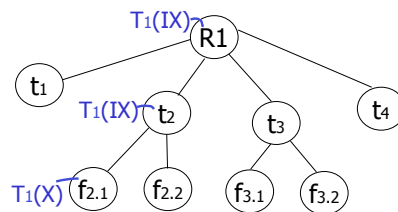
Notes 14 - Concurrency Control

129

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Exercise:

- Can T2 access object f2.2 in X mode? What locks will T2 get?



CS 525



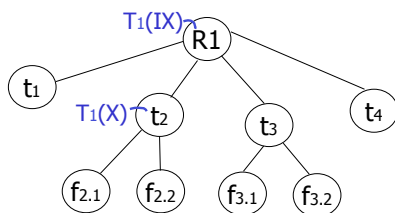
Notes 14 - Concurrency Control

130

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Exercise:

- Can T2 access object f2.2 in X mode? What locks will T2 get?



CS 525



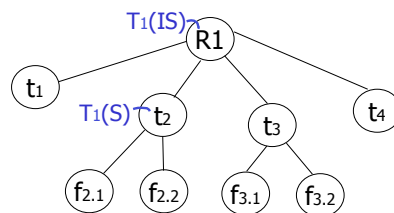
Notes 14 - Concurrency Control

131

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Exercise:

- Can T2 access object f3.1 in X mode? What locks will T2 get?



CS 525



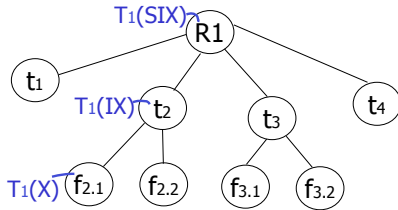
Notes 14 - Concurrency Control

132

IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

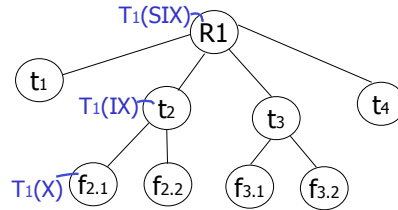
Exercise:

- Can T2 access object f2.2 in S mode?
What locks will T2 get?

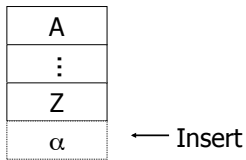


Exercise:

- Can T2 access object f2.2 in X mode?
What locks will T2 get?



Insert + delete operations



Modifications to locking rules:

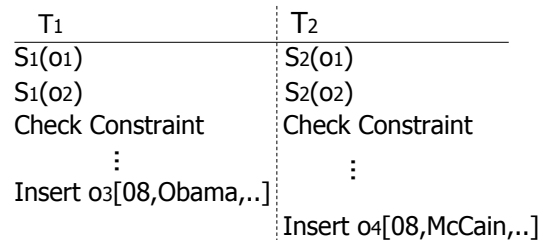
- (1) Get exclusive lock on A before deleting A
- (2) At insert A operation by T_i, T_i is given exclusive lock on A

Still have a problem: **Phantoms**

Example: relation R (E#,name,...)
constraint: E# is key
use tuple locking

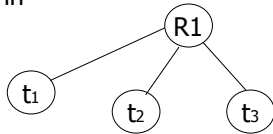
R	E#	Name	...
o1	55	Smith	
o2	75	Jones	

T₁: Insert <08,Obama,...> into R
T₂: Insert <08,McCain,...> into R

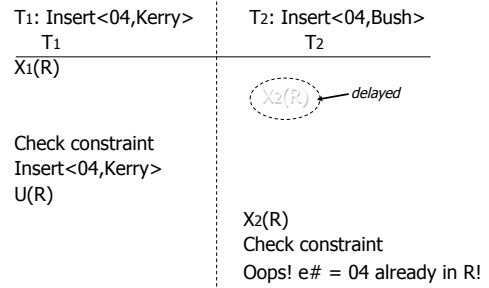


Solution

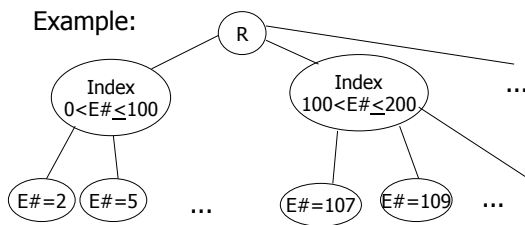
- Use multiple granularity tree
- Before insert of node Q, lock parent(Q) in X mode



Back to example



Instead of using R, can use index on R:



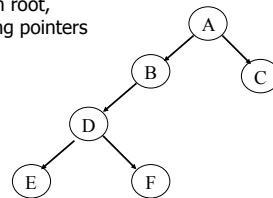
- This approach can be generalized to multiple indexes...

Next:

- Tree-based concurrency control
- Validation concurrency control

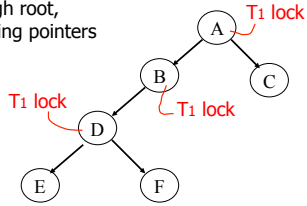
Example

- all objects accessed through root, following pointers



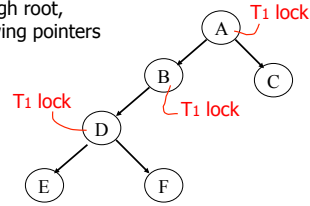
Example

- all objects accessed through root, following pointers



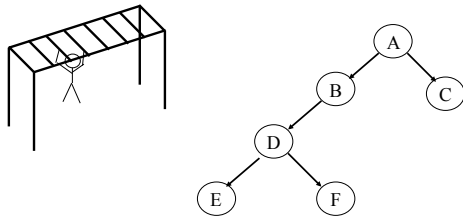
Example

- all objects accessed through root, following pointers

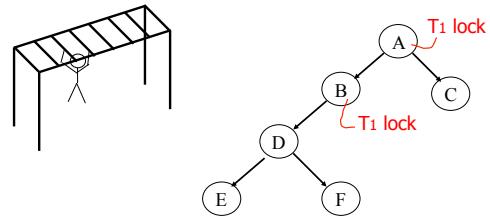


← can we release A lock if we no longer need A??

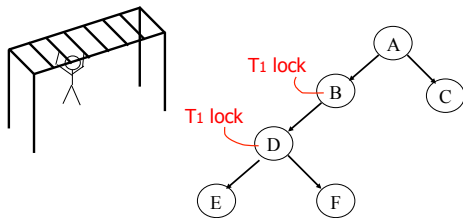
Idea: traverse like “Monkey Bars”



Idea: traverse like “Monkey Bars”

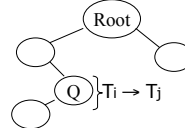


Idea: traverse like “Monkey Bars”



Why does this work?

- Assume all T_i start at root; exclusive lock
- $T_i \rightarrow T_j \Rightarrow T_i$ locks root before T_j

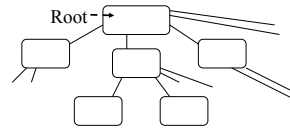


- Actually works if we don't always start at root

Rules: tree protocol (exclusive locks)

- (1) First lock by T_i may be on any item
- (2) After that, item Q can be locked by T_i only if $\text{parent}(Q)$ locked by T_i
- (3) Items may be unlocked at any time
- (4) After T_i unlocks Q , it cannot relock Q

- Tree-like protocols are used typically for B-tree concurrency control



E.g., during insert, do not release parent lock, until you are certain child does not have to split

CS 525



Notes 14 - Concurrency Control

151 IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

CS 525

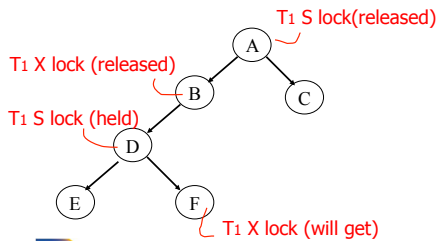


Notes 14 - Concurrency Control

152 IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Tree Protocol with Shared Locks

- Rules for shared & exclusive locks?



CS 525

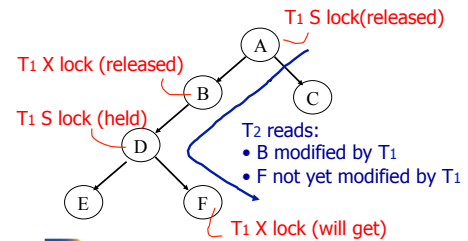


Notes 14 - Concurrency Control

153 IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Tree Protocol with Shared Locks

- Rules for shared & exclusive locks?



CS 525



Notes 14 - Concurrency Control

154 IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Tree Protocol with Shared Locks

- Need more restrictive protocol
- Will this work??
 - Once T_1 locks one object in X mode, all further locks down the tree must be in X mode

Deadlocks (again)

- Before we assumed that we are able to detect deadlocks and resolve them
- Now two options
 - (1) Deadlock detection (and resolving)
 - (2) Deadlock prevention

CS 525



Notes 14 - Concurrency Control

155 IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

CS 525

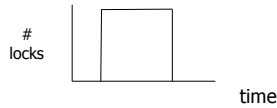


Notes 14 - Concurrency Control

156 IIT College of Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Deadlock Prevention

- Option 1:
 - 2PL + transaction has to acquire all locks at transaction start following a global order



Deadlock Prevention

- Option 1:
 - Long lock durations ☹
 - Transaction has to know upfront what data items it will access ☹
 - E.g.,
UPDATE R SET a = a + 1 WHERE b < 15
 - We don't know what tuples are in R!

Deadlock Prevention

- Option 2:
 - Define some global order of data items O
 - Transactions have to acquire locks according to this order
- Example ($X < Y < Z$)
 - $I_1(X), I_1(Z)$ (OK)
 - $I_1(Y), I_1(X)$ (NOT OK)

Deadlock Prevention

- Option 2:
 - Accessed data items have to be known upfront ☹
 - or access to data has to follow the order ☹

Deadlock Prevention

- Option 3 (**Preemption**)
 - Roll-back transactions that wait for locks under certain conditions
 - 3 a) **wait-die**
 - Assign timestamp to each transaction
 - If transaction T_i waits for T_j to release a lock
 - Timestamp $T_i < T_j$ -> wait
 - Timestamp $T_i > T_j$ -> roll-back T_i

Deadlock Prevention

- Option 3 (**Preemption**)
 - Roll-back transactions that wait for locks under certain conditions
 - 3 a) **wound-wait**
 - Assign timestamp to each transaction
 - If transaction T_i waits for T_j to release a lock
 - Timestamp $T_i < T_j$ -> roll-back T_j
 - Timestamp $T_i > T_j$ -> wait

Deadlock Prevention

- Option 3:
 - Additional transaction roll-backs ☹

Timeout-based Scheme

- Option 4:
 - After waiting for a lock longer than X, a transaction is rolled back

Timeout-based Scheme

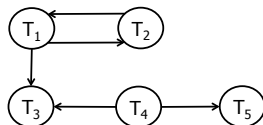
- Option 4:
 - Simple scheme ☺
 - Hard to find a good value of X
 - To high: long wait times for a transaction before it gets eventually aborted
 - To low: too many transaction that are not deadlock get aborted

Deadlock Detection and Resolution

- Data structure to detect deadlocks: **wait-for** graph
 - One node for each transaction
 - Edge $T_i \rightarrow T_j$ if T_i is waiting for T_j
 - Cycle \rightarrow Deadlock
 - Abort one of the transaction in cycle to resolve deadlock

Deadlock Detection and Resolution

- When do we run the detection?
- How to choose the victim?



Optimistic Concurrency Control: Validation

Transactions have 3 phases:

- (1) Read
 - all DB values read
 - writes to temporary storage
 - no locking
- (2) Validate
 - check if schedule so far is serializable
- (3) Write
 - if validate ok, write to DB

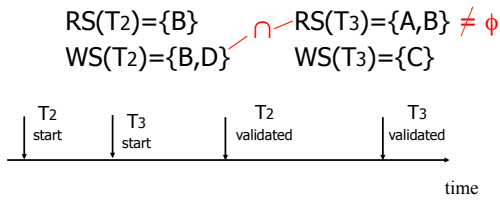
Key idea

- Make validation atomic
- If T_1, T_2, T_3, \dots is validation order, then resulting schedule will be conflict equivalent to $S_s = T_1 T_2 T_3 \dots$

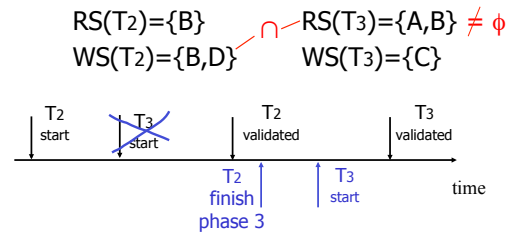
To implement validation, system keeps two sets:

- FIN = transactions that have finished phase 3 (and are all done)
- VAL = transactions that have successfully finished phase 2 (validation)

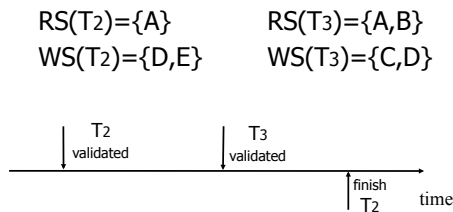
Example of what validation must prevent:



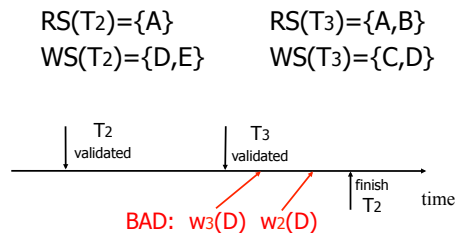
Example of what validation must prevent:



Another thing validation must prevent:

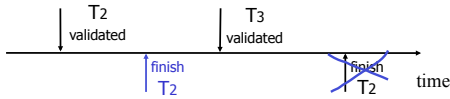


Another thing validation must prevent:



Another thing validation must ^{allow} prevent:

$RS(T_2) = \{A\}$ $RS(T_3) = \{A, B\}$
 $WS(T_2) = \{D, E\}$ $WS(T_3) = \{C, D\}$



Validation rules for T_j :

(1) When T_j starts phase 1:

$ignore(T_j) \leftarrow FIN$

(2) at T_j Validation:

if check (T_j) then

[$VAL \leftarrow VAL \cup \{T_j\}$;

do write phase;

$FIN \leftarrow FIN \cup \{T_j\}$]

Check (T_j):

For $T_i \in VAL - IGNORE(T_j)$ DO

IF [$WS(T_i) \cap RS(T_j) \neq \emptyset$ OR

$T_i \notin FIN$] THEN RETURN false;

RETURN true;

Check (T_j):

For $T_i \in VAL - IGNORE(T_j)$ DO

IF [$WS(T_i) \cap RS(T_j) \neq \emptyset$ OR

$T_i \notin FIN$] THEN RETURN false;

RETURN true;

Is this check too restrictive ?

Improving Check(T_j)

For $T_i \in VAL - IGNORE(T_j)$ DO

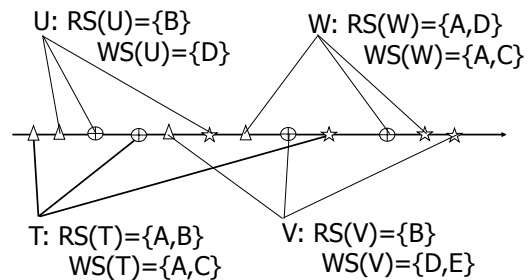
IF [$WS(T_i) \cap RS(T_j) \neq \emptyset$ OR

$(T_i \notin FIN \text{ AND } WS(T_i) \cap WS(T_j) \neq \emptyset)$

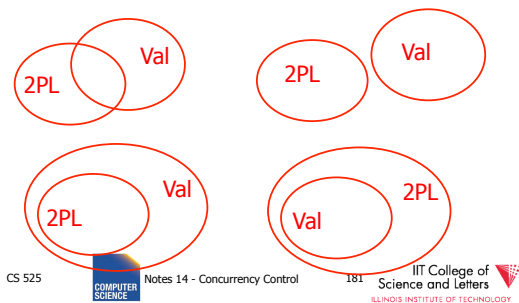
THEN RETURN false;

RETURN true;

Exercise:



Is Validation = 2PL?



S2: $w2(y) \ w1(x) \ w2(x)$

- S2 can be achieved with 2PL:
 $l2(y) \ w2(y) \ l1(x) \ w1(x) \ u1(x) \ l2(x) \ w2(x) \ u2(y) \ u2(x)$
- S2 cannot be achieved by validation:
 The validation point of T2, val2 must occur before $w2(y)$ since transactions do not write to the database until after validation. Because of the conflict on x, $val1 < val2$, so we must have something like
 $S2: \ val1 \ val2 \ w2(y) \ w1(x) \ w2(x)$
 With the validation protocol, the writes of T2 should not start until T1 is all done with its writes, which is not the case.

Validation subset of 2PL?

- Possible proof (Check!):
 - Let S be validation schedule
 - For each T in S insert lock/unlocks, get S' :
 - At T start: request read locks for all of RS(T)
 - At T validation: request write locks for WS(T); release read locks for read-only objects
 - At T end: release all write locks
 - Clearly transactions well-formed and 2PL
 - Must show S' is legal (next page)

- Say S' not legal:
 $S' : \dots \ l1(x) \ w2(x) \ r1(x) \ val1 \ u2(x) \dots$
 - At val1: T2 not in Ignore(T1); T2 in VAL
 - T1 does not validate: $WS(T2) \cap \ RS(T1) \neq \emptyset$
 - contradiction!
- Say S' not legal:
 $S' : \dots \ val1 \ l1(x) \ w2(x) \ w1(x) \ u2(x) \dots$
 - Say T2 validates first (proof similar in other case)
 - At val1: T2 not in Ignore(T1); T2 in VAL
 - T1 does not validate:
 $T2 \notin \text{FIN} \ \text{AND} \ WS(T1) \cap \ WS(T2) \neq \emptyset$
 - contradiction!

Validation (also called **optimistic concurrency control**) is useful in some cases:

- Conflicts rare
- System resources plentiful
- Have real time constraints

Multiversioning Concurrency Control (MVCC)

- Keep old versions of data item and use this to increase concurrency
- Each write creates a new version of the written data item
- Use version numbers of timestamps to identify versions

Multiversioning Concurrency Control (MVCC)

- **Different transactions** operate over **different versions** of data items
- -> readers never have to wait for writers
- -> great for combined workloads
 - **OLTP** workload (writes, only access small number of tuples, short)
 - **OLAP** workload (reads, access large portions of database, long running)

CS 525  Notes 14 - Concurrency Control 187 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

MVCC schemes

- MVCC timestamp ordering
- MVCC 2PL
- Snapshot isolation (SI)
 - We will only cover this one

CS 525  Notes 14 - Concurrency Control 188 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

Snapshot Isolation (SI)

- Each transaction **T** is assigned a timestamp **S(T)** when it starts
- Each write creates a new data item version timestamped with the current timestamp
- When a transaction commits, then the latest versions created by the transaction get a timestamp **C(T)** as of the commit

CS 525  Notes 14 - Concurrency Control 189 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

Snapshot Isolation (SI)

- Under snapshot isolation each transaction **T** sees a consistent snapshot of the database as of **S(T)**
 - It only sees data item versions of transactions that committed before **T** started
 - It also sees its own changes

CS 525  Notes 14 - Concurrency Control 190 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

First Updater Wins Rule (FUW)

- Two transactions T_i and T_j may update the same data item **A**
 - To avoid lost updates only one of the two can be safely committed
- **First Updater Wins Rules**
 - The transaction that updated **A** first is allowed to commit
 - The other transaction is aborted

CS 525  Notes 14 - Concurrency Control 191 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

First Committer Wins Rule (FCW)

- Two transactions T_i and T_j may update the same data item **A**
 - To avoid lost updates only one of the two can be safely committed
- **First Committer Wins Rules**
 - The transaction that attempts to commit first is allowed to commit
 - The other transaction is aborted

CS 525  Notes 14 - Concurrency Control 192 IIT College of Science and Letters ILLINOIS INSTITUTE OF TECHNOLOGY

@Silberschatz, Korth and Sudarshan

	X	Y	Z		T1	T2	T3
	0	1	5	1	W(Y := 1)		
Update not visible outside of T1	0			2	Commit		
Update becomes visible to future transactions	0			3			
				4		Start	
				5		R(X) → 0	
				6		R(Y) → 1	
	2			7			
	2			8			
			3	9			W(X:=2)
Concurrent updates not visible			3	10			W(Z:=3)
				11		R(Z) → 5	
				12		R(Y) → 1	
Not first-committer of X	3			13		W(X:=3)	
Serialization error, T2 is rolled back				14		Commit-Req	
						Abort	

CS 525



Notes 14 - Concurrency Control

193



Why does that work?

- Since all transactions see a consistent snapshot and their changes are only made "public" once they commit
 - It looks like the transactions have been executed in the order of their commits*

* Recall the writes to the same data item are disallowed for concurrent transactions

CS 525



Notes 14 - Concurrency Control

194



Is that serializable?

- Almost ;-)
- There is still one type of conflict which cannot occur in serializable schedules called **write-skew**

CS 525



Notes 14 - Concurrency Control

195



Write Skew

- Consider two data items A and B
 - A = 5, B = 5
- Concurrent Transactions T1 and T2
 - T1: A = A + B
 - T2: B = A + B
- Final result under SI
 - A = 10, B = 10

CS 525



Notes 14 - Concurrency Control

196



Write Skew

- Consider serial schedules:
 - T1, T2: A=10, B=15
 - T2, T1: A=15, B=10
- What is the problem
 - Under SI both T1 and T2 do not see each others changes
 - In any serial schedule one of the two would see the others changes

CS 525



Notes 14 - Concurrency Control

197



Example: Oracle

- Tuples are updated in place
- Old versions in separate ROLLBACK segment
 - GC once nobody needs them anymore
- How to implement the FCW or FUW?
 - Oracle uses write locks to block concurrent writes
 - Transaction waiting for a write lock aborts if transaction holding the lock commits

CS 525



Notes 14 - Concurrency Control

198



SI Discussion

- **Advantages**
 - Readers and writers do not block each other
 - If we do not GC old row versions we can go back to previous versions of the database -> Time travel
 - E.g., show me the customer table as it was yesterday
- **Disadvantages**
 - Storage overhead to keep old row versions
 - GC overhead
 - Not strictly serializable

CS 525



Notes 14 - Concurrency Control

199

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY

Summary

Have studied CC mechanisms used in practice

- 2 PL variants
- Multiple lock granularity
- Deadlocks
- Tree (index) protocols
- Optimistic CC (Validation)
- Multiversioning Concurrency Control (MVCC)

CS 525



Notes 14 - Concurrency Control

200

IIT College of
Science and Letters
ILLINOIS INSTITUTE OF TECHNOLOGY