Announcements

- Welcome to CSc 460 - Database Design!
- Class Information Sources:
  - D2L (d2l.arizona.edu/d2l/home/1186894)
  - Web (u.arizona.edu/~mccann/classes/460)
  - Piazza (piazza.com/class/l5sr9c7qjh22bi)
- Class Modality: In-person. Per University policy, this class is available only in the modality in the catalog.
  - Masking is not required in class. Please respect the choices of your fellow students.
  - Lectures will be recorded (when the technology is working) and will be available via D2L (in the Content section, under Lecture Recordings).

Continues . . .

Announcements (continued!)

- Handouts: (1) Summary of the Class Syllabus
  (2) Program #1
  (3) Program Style Requirements
  - We will cover these (inc. the full syllabus) today
- Exam dates are already scheduled (see the syllabus' Topic Schedule). Please schedule your job interviews, weddings, espionage trials, etc., accordingly.
- The Fall CS Career Fairs are coming soon:
  - Virtual: Sept 15th
  - In-person: Sept 16th

  } See Arthur's Aug 16th email
- Any questions before we get started?

Announcements

- Program #1 is now officially assigned
  - Part A is due in one week
    - If you need to fix it later, please do! (Students often discover issues w/ Part A while working on Part B.)
    - We <u>will</u> 'grade' your Part A submission, and deduct late days as appropriate, so please take the Part A due date seriously.
    - One correction so far: The State field is also a string field.

Topic 1:

Databases and Database Management Systems

Definitions of 'Database'

- "A shared collection of logically related data and its description, designed to meet the information needs of an organization." (Connolly/Begg, 6/e)
- "... a collection of persistent data that is used by the application systems of some given enterprise" (Date, 7/e)
- "A collection of related data" (Elmasri/Navathe, 5/e)
- "[A] collection of records kept for a common purpose" (O'Neil[2], 2/e)
- "A collection of data, typically describing the activities of one or more related organizations" (Ramakrishnan/Gehrke, 3/e)
- "A collection of related records stored with a minimum of redundancy that many users can share simultaneously" (Shepherd)
- "[A] collection of [interrelated] data ... contain[ing] information relevant to an enterprise" (Silberschatz/Korth/Sudarshan, 4/e)

Common Definition Themes

Three very popular ideas in the definitions:

- **Collection**
- **(Inter) Related**
- **Data / Records**

Other ideas of note:

- Enterprise / Organization / Common Purpose
- Sharing
- Minimized Redundancy
- Persistence

Questions:

- Is your cell phone's contact list a database? yes
- Is your wallet a database? yes
- Where is the software?

Definitions of 'Database Management System'

DBMS

- "A software system that enables users to define, create, maintain, and control access to the database." (Connolly/Begg, 6/e, p. 16)
- "A computerized record–keeping system" (Date, 7/e, p. 2)
- "A collection of programs that enables users to create and maintain a database" (Elmasri/Navathe, 5/e, p. 5)
- "A program product for keeping computerized records about an enterprise" (O'Neil/O'Neil, 2/e, p. 1)
- "Software designed to assist in maintaining and utilizing large collections of data" (Ramakrishnan/Gehrke, 3/e, p. 4)
- "Cost–effective methods for storing, organizing, retrieving, and managing data" (Shepherd)
- "A collection of interrelated data and a set of programs to access those data" (Silberschatz/Korth/Sudarshan, 4/e, p. 1)

DBMS Components

1. The Database
2. Administration – Database Administrator (DBA)
   - handle maintenance, performance, security, user education, ...
   - regular supervision is needed!
3. Application Programs
   - from text queries to web interfaces
   - there are standard programming interfaces (e.g. JDBC)
4. Hardware
   - often a dedicated cluster of servers

Why use a DBMS? (1 / 3)

1. Data Sharing
   - one common location vs. many individual copies
   - helps maintain data consistency between copies

2. Redundancy Control
   - intra-file redundancy managed by Data Normalization
   - Distributed DBMSes intentionally copy files to maintain accessibility

3. Centralized Control
   - DBAs provide experienced & benevolent oversight
   - Allows users to be users

Why use a DBMS? (2 / 3)

(4) Data Integrity
- Ex: June 31st
- Most integrity info is domain-specific
- DBMSes support user-defined integrity rules

(5) Data Security
- DBMSes can flexibly limit access to users
- Active management by DBA > user management

(6) Views
- are automatically managed presentations of the database
- provides clarity, convenience, & some security

Why use a DBMS? (3 / 3)

(7) Data Independence
- Idea: User's shouldn't know how data is organized within the DBMS
- 2 types:
  - (a) Logical Data Independence (LDI)
    - Conceptual Schema - describes the data and its relationships
    - LDI allows schema to change w/o applications changing
    - Difficult to achieve
  - (b) Physical Data Independence (PDI)
    - PDI allows physical storage to change w/o applications changing (e.g., create index)
    - More easily achieved than LDI

Disadvantages of DBMSes

1. DB design is complex
   - matching design to desired function is not trivial
2. Cost
   - organizational licenses are expensive
   - DBAs need to be paid
   - Servers are not cheap
3. Availability
   - A DBMS is a potential "single" point of failure
   - Making backups is important!
4. Speed v. Flexibility
   - Users want high performance, but DBMSes are designed for general purpose use.

Announcements

- Program #1:
  - Part A is due Wednesday. We're looking for:
    - your Prog1A.java file that is...
    - a good attempt at creating the binary file, with...
    - good documentation & coding style
  - Follow the instructions in the Hand In section of the assignment handout to submit your code.
  - Part B is due next Wednesday.
- Our office hours are set – see Piazza and D2L for the schedule. We've started holding them today.

Data Languages

Four types of manipulation, four languages, all within SQL:

- Data Description Language (DDL)
  - turns the conceptual schema into a physical database description
- Data Manipulation Language (DML)
  - insertion / deletion of data
- Data Control Language (DCL)
  - security
- Query Language
  - read-only, to answer questions

Topic 2:

Database Management System Architectures

What is an 'Architecture' of a DBMS?

Definition: Schema

The overall description of a database
– at varying granularities

Definition: Architecture

A description of DBMS components and their interconnections.

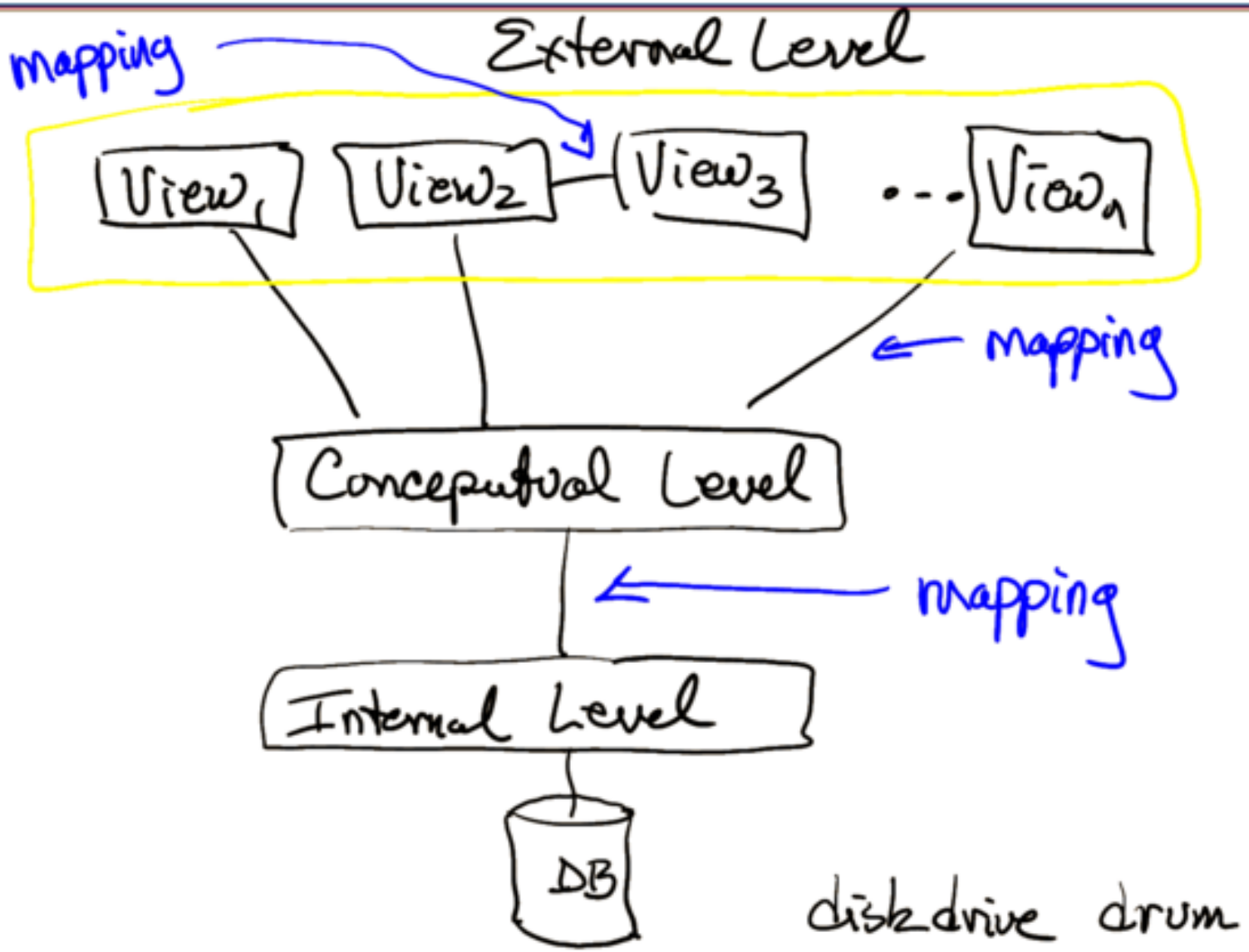
Notes:
- the schema is part of the architecture
- often, the terms are used interchangeably
- I have not seen a text define "architecture" ↑ DB

The ANSI/SPARC Architecture (1 / 4): Background

a.k.a. Three-Level Schema

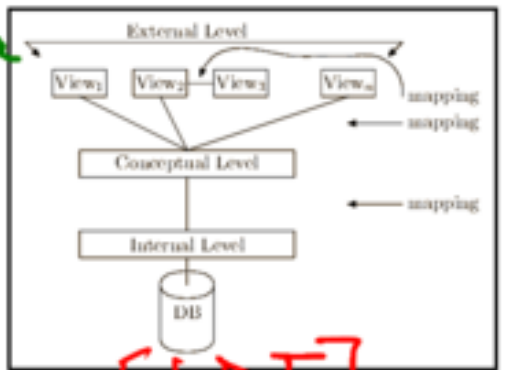
- A product of the Standards Planning and Requirements Committee (SPARC) of the American National Standards Institute (ANSI)
- Never formally adopted as an ANSI or International Standards Organization (ISO) standard, but still very influential
- Created to standardize terms and concepts surrounding DBs and DBMSes
- Goals:
  - Allow for multiple views of the data to satisfy a range of users
  - Allow for a physical (disk-level) description of the database
  - Provide an abstraction layer to separate the two

The ANSI/SPARC Architecture (2 / 4): The Diagram



The ANSI/SPARC Architecture (3 / 4): The Levels

- External Level – described by External Schema
  - defines end-user perspective on the DB



- Conceptual Level – desc. by a Conceptual Schema [CDI]
  - defines field groupings, data relationships, etc.
  - abstract from physical considerations.

- Internal Level – described by an Internal Schema [PDI]
  - defines record sizes, indices, field representations, record ordering, etc.
  - hidden from applications & users

The ANSI/SPARC Architecture (4 / 4): The Mappings

The interfaces between the levels are known as *mappings*.

- External – External Mapping
  - often easier to tweak an existing view
  - usually more efficient to base views on Conceptual Level.
- External – Conceptual Mapping
  - Allows field renamings & arrangements for the user's benefit.
  - provides Logical Data Independence (LDI)
- Conceptual – Internal Mapping
  - Converts logical structure to physical reps.
  - provides Physical Data Indep. (PDI)

Client - Server Architectures (1 / 3): Background

- Originally: DBMSes were built with a centralized architecture.
  - All components (OS, DBMS, compilers, etc.) on one computer
  - “All or nothing” with respect to failures
  - Often a performance bottleneck
- Decentralization became feasible when:
  - Computers became less expensive and more powerful
  - Broadband networking became commonplace

Client - Server Architectures (2 / 3): Two-Tier

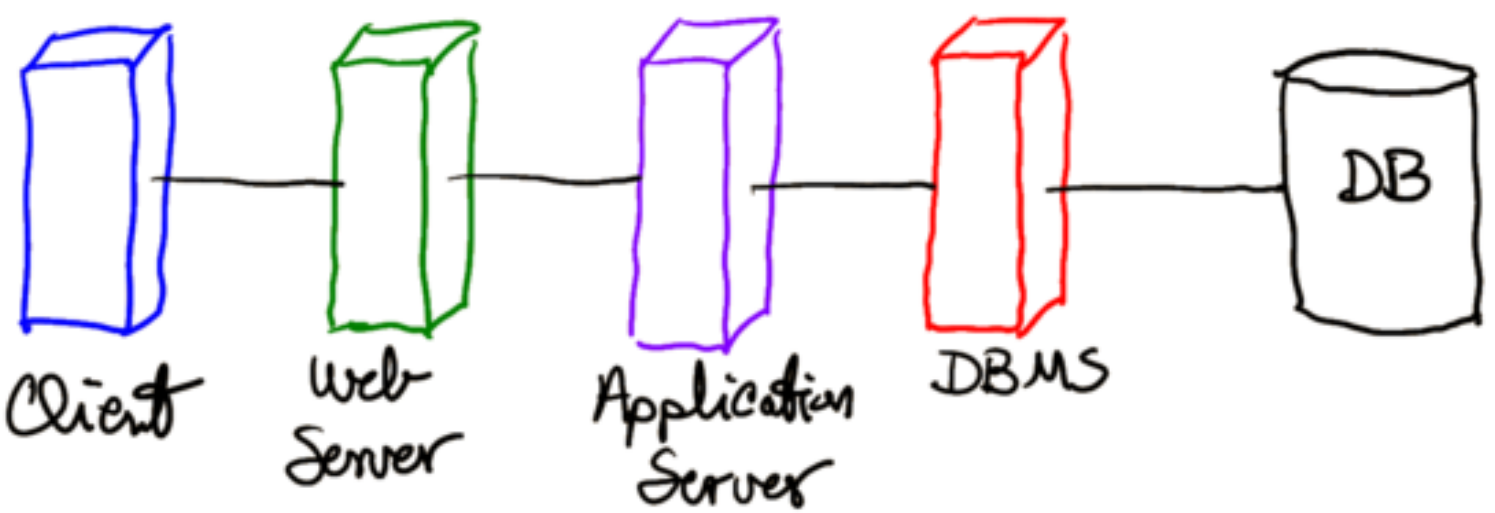
- One possible division of services:
  1. The client – presents info to user, runs applications that get data from --
  2. The server – fields requests from clients, runs the DBMS
- Lots of variations
  - #s of clients & #s of servers
  - parts of DBMS can be executed/stored by clients.

Client - Server Architectures (3 / 3): Multi-Tier

Why add more tiers?

Add new capabilities, spread out old ones, ...

Example of a Four-Tier architecture:



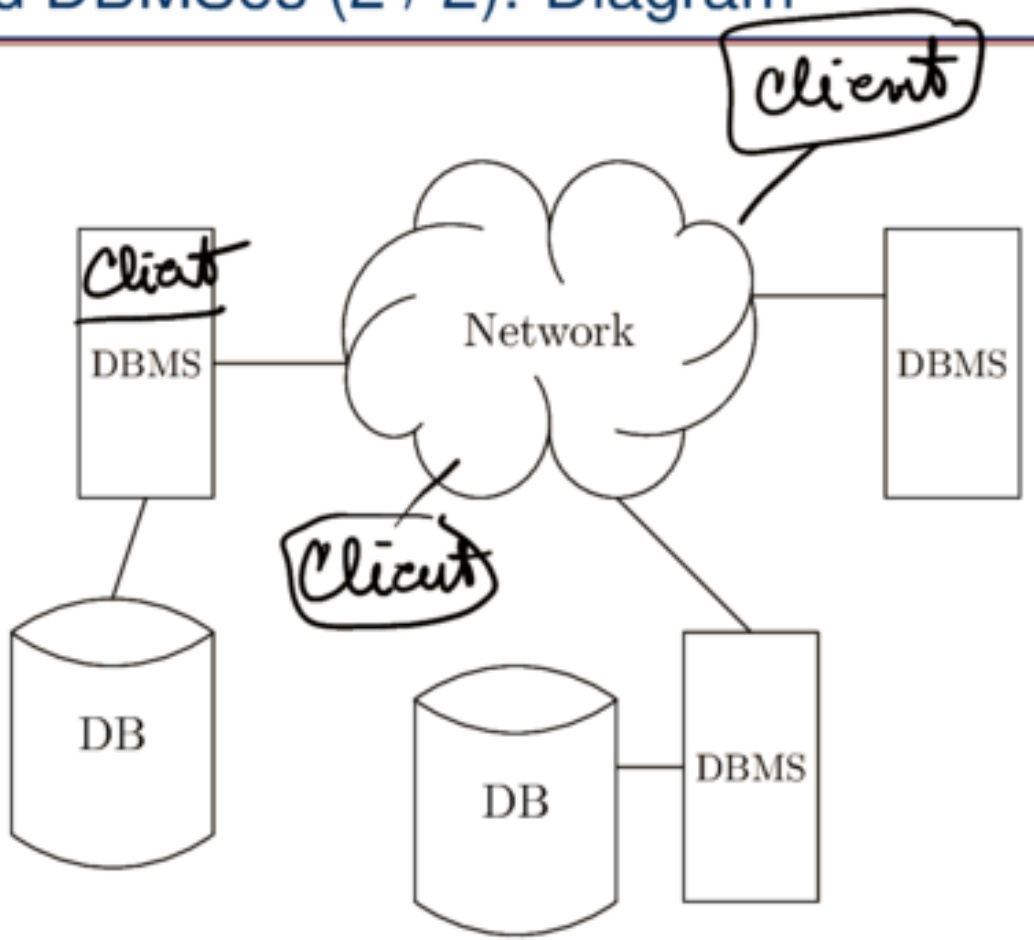
Distributed DBMSes (1 / 2): Motivation

DDBMS

- A single DBMS server (with its single DB) is a single point of failure
- Solution: A DBMS can be operated by several servers.
  - Each server has all, some, or none of the DB stored locally

    (replication is permitted for performance and reliability)
  - DDBMS sites communicate to handle nearly all tasks
  - Goal: Be completely transparent to the users
- Again, details are beyond the scope of this course

Distributed DBMSes (2 / 2): Diagram



Extra Slides

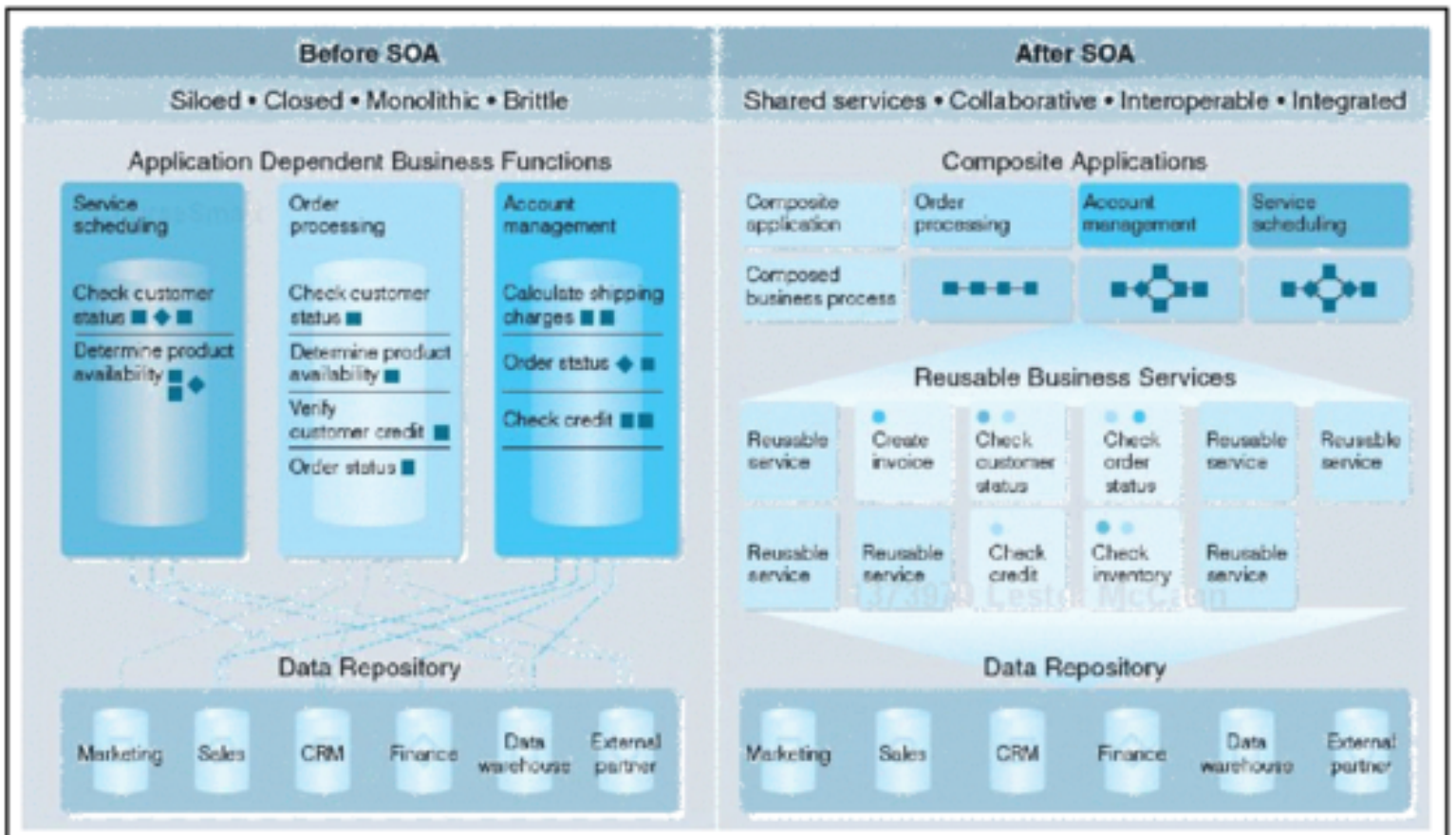
The remaining slides in this topic are some that I no longer cover in class. I won't ask about them on an exam, but they could be referenced on a homework.

Service-Oriented Architectures (1 / 3): Motivation

- SOA is a software design technique:
  - Apps are built using pre-written service modules

    E.g., a data visualization module
  - Modules are located & accessed via a common interface
- Goal is to be flexible with the adoption of new business processes
- A web service is an interface used by service modules
  - That is, it can be a component of an SOA.
- Further details are beyond the scope of this course

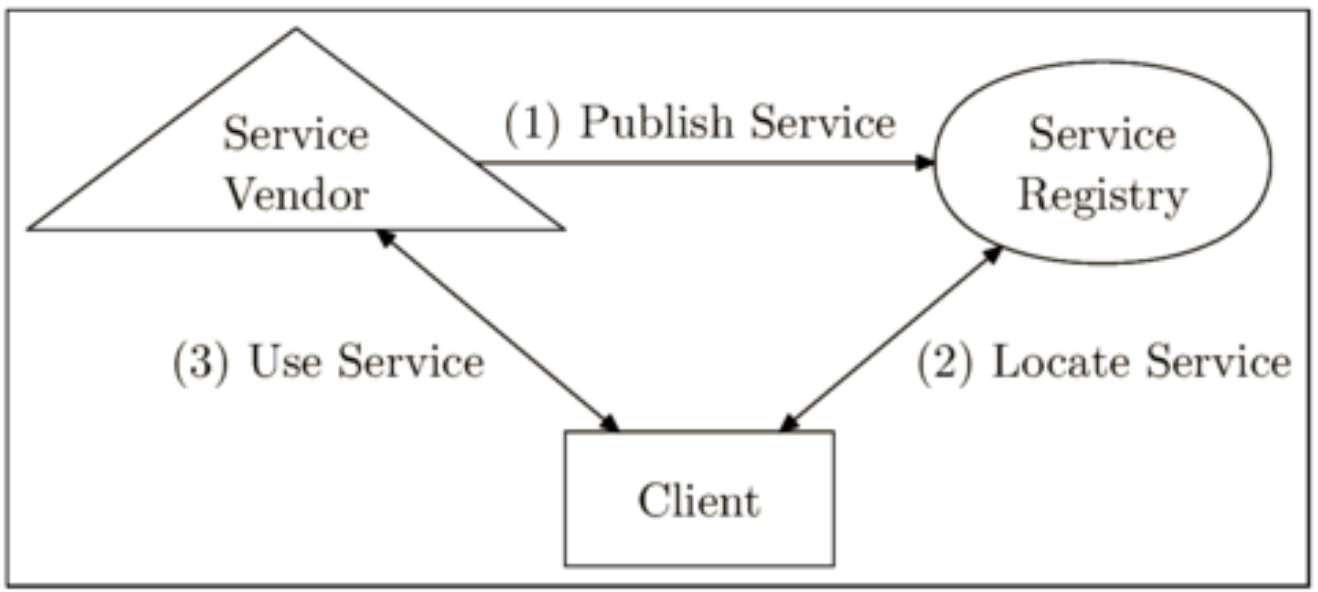
Service-Oriented Architectures (2 / 3): Before/After



Credit: Connolly/Begg, "Database Systems," 6/e, p. 72.

Service-Oriented Architectures (3 / 3): Accessing

Advertising, Finding and Using a Service:



One example: Web browser plug-ins.

(This approach is typical for web services, too.)

Additional DB–Related Architectures

- Web Services
  - Ex: Stock Quotations; Google Docs

  Two types:

  (a) Simple Object Access Protocol (SOAP)-based
  – Typically uses XML

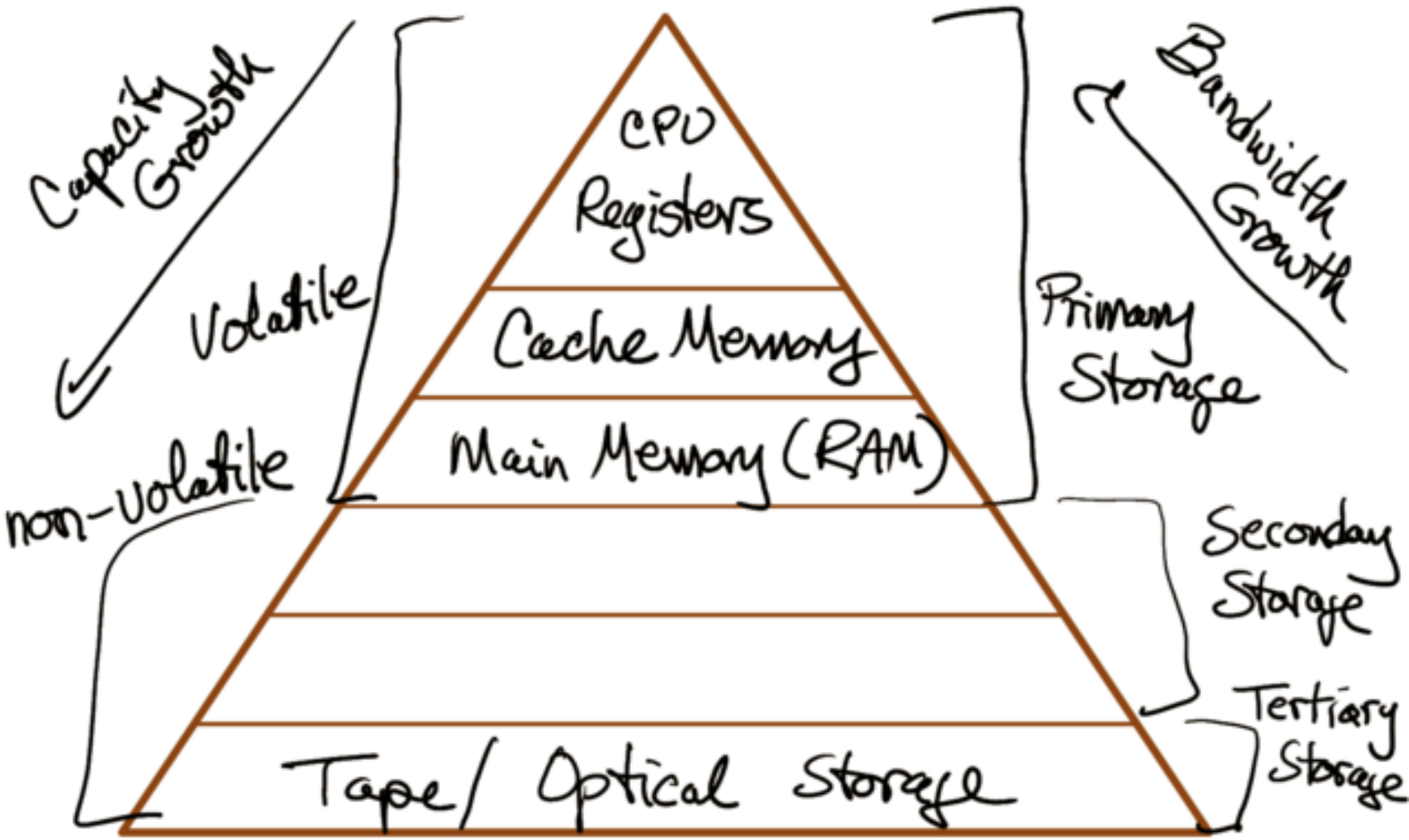
  (b) RESTful (Representational State Transfer) – stateless
  – Ex: HTTP

- Data Warehouses
  - Support decision–making
- Cloud Computing
  - Provides dynamic resource provisioning (of DBMSes!)

Topic 3:

Files and Indexing

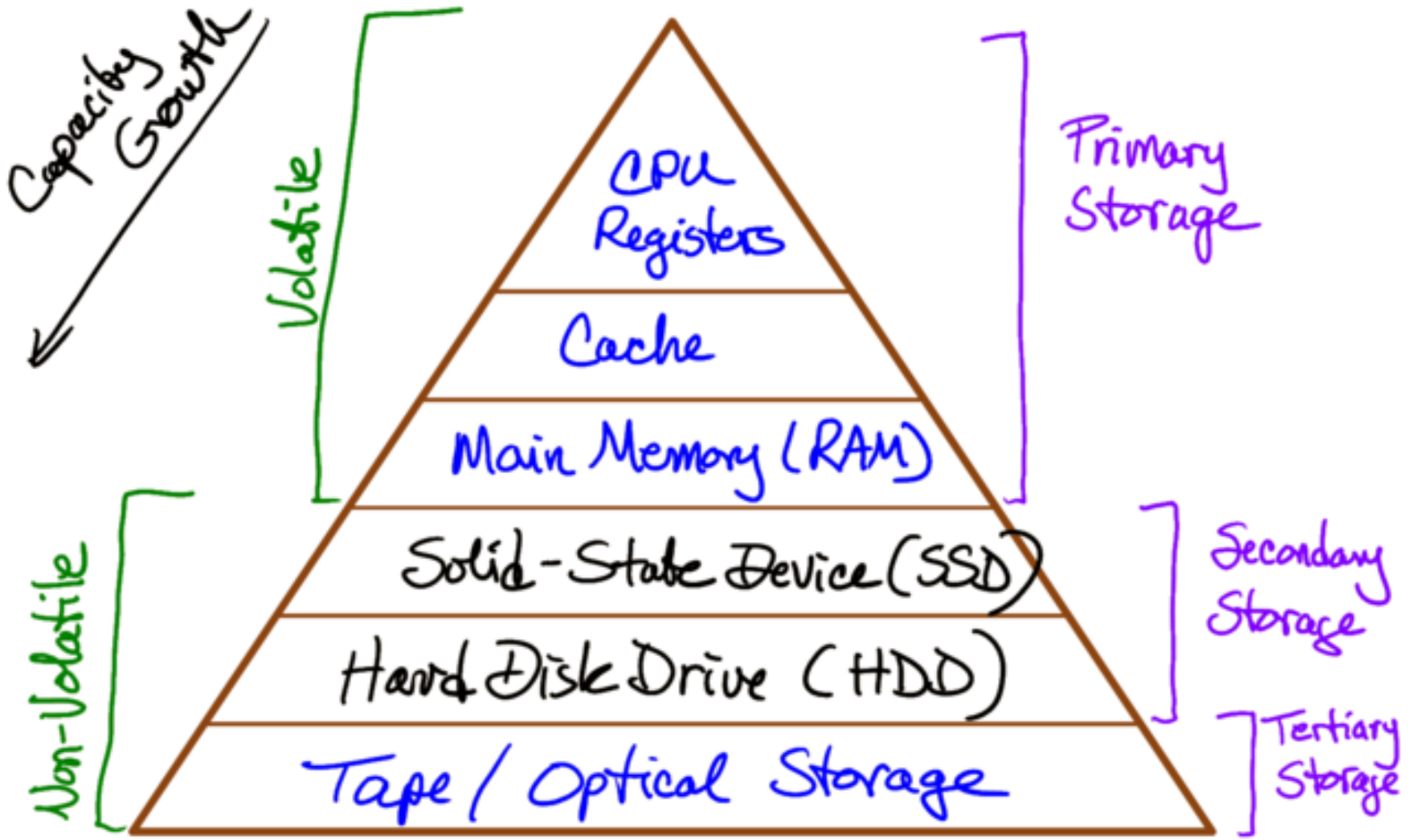
The Storage Pyramid



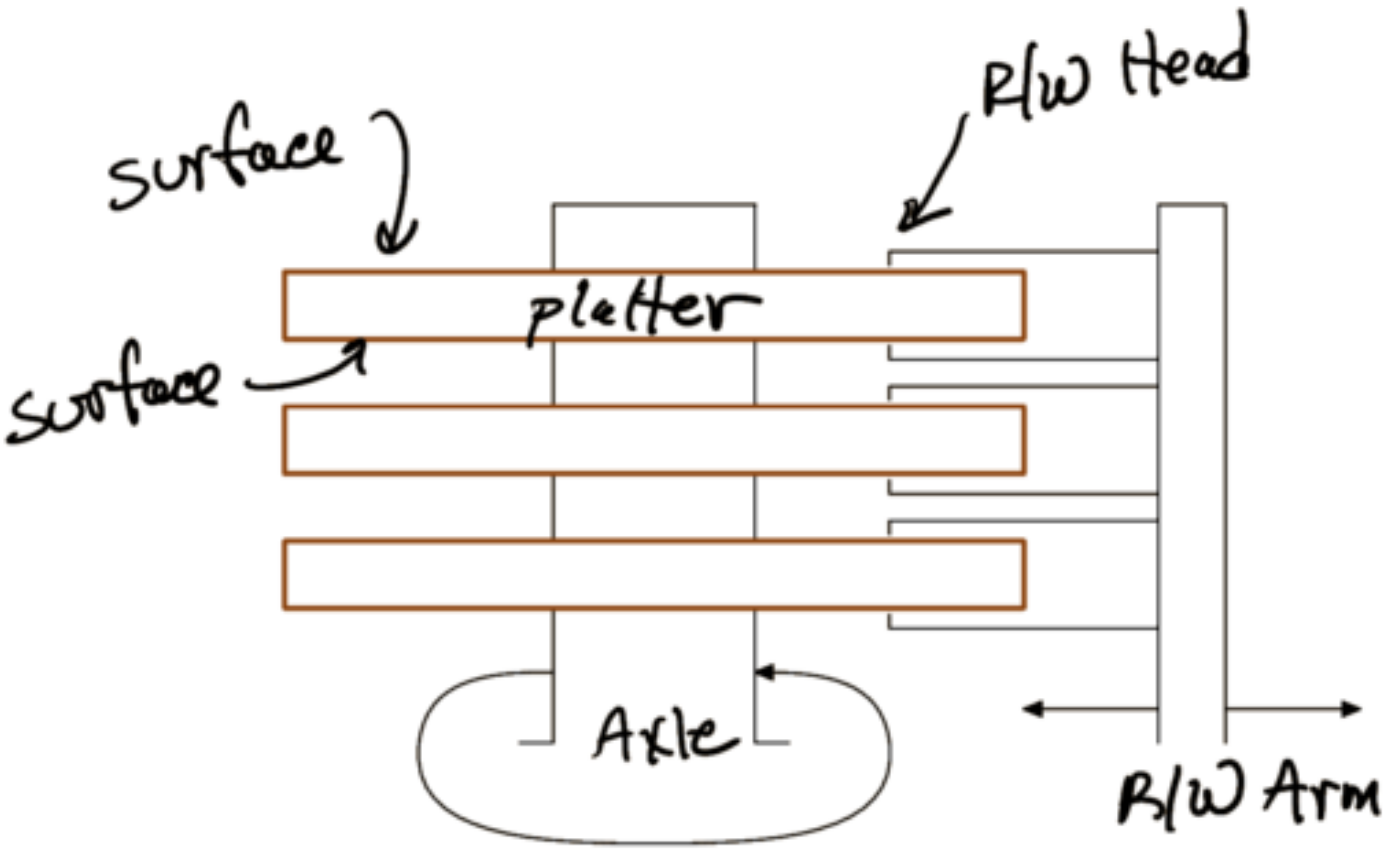
Announcements

- Program 1:
  - Part A "first draft" is due now
    - TAs will only check submission for documentation and style, not for correct operation
  - Complete assignment is due in one week
    - Submit both Part B and Part A, to be sure that they work together correctly
- Program #2 will be assigned next Wednesday
- No classes Monday - Labor Day!
  - Our only day off all semester...
- CS Tutor Center has quite a bit of coverage for 460
- "Old Storage Device Show and Tell" available w/ lecture recordings in DZL.

The Storage Pyramid

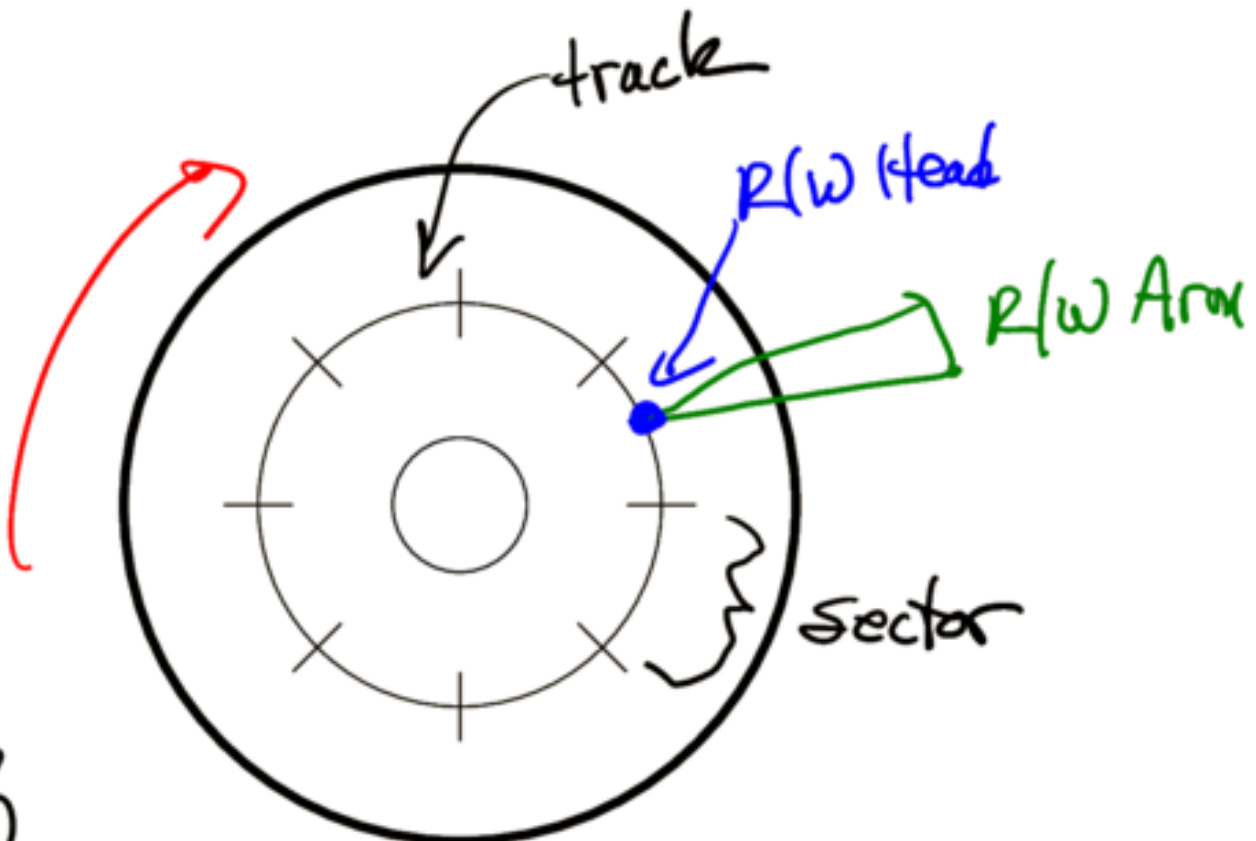


Hard Drive Physical Characteristics (1 / 2)



Side View

Hard Drive Physical Characteristics (2 / 2)



Top View (one surface)

A block is a group of sectors

⇒ The basic unit of HDD storage

Sources of Read / Write Delay

The three major sources of delay (in descending order):

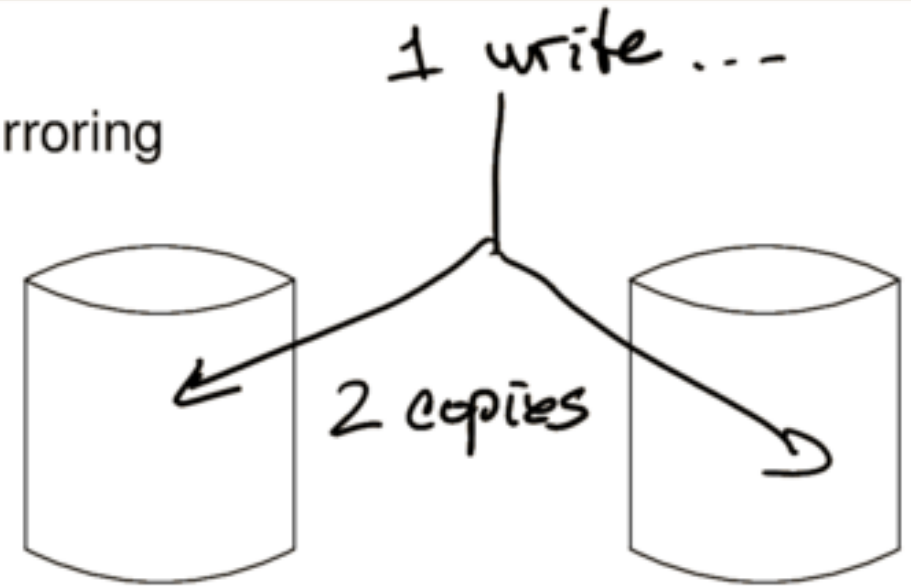
1. Seek Time
   - move R/W head to appropriate track
2. (Rotational) Latency
   - wait for block to rotate to R/W Head
3. Transfer Time
   - read/write data from/to surface.

Western Digital 3.5” Hard Drive Specs

	WD450AA (9/2000)	WD1001FALS (7/2009)	WD4003FZEX (7/2015)	DC HC550 (8/2020)
Size (GB)	45	1,000	4,000	18,000
Platters & Heads	3 & 6	3 & 6	5 & 10	9 & 18
Bytes per Sector	512	512	512	512 / 4096
Sectors per Surface	14,655,144	325,587,528	781,403,717	??
Rotations (RPM)	5400	7200	7200	7200
R/W Seeks (ms)	9.5 / 13.4	?? / ??	?? / ??	?? / ??
Latency (ms)	5.4	4.2	??	4.16
Cache (MB)	2	32	64	512
Buffer to Host (MB/s max.)	66.6	3000.0	6000.0	600.0
[Power] Read / Write (W)	6.2	8.4	9.5	6.5
Idle (W)	6.2	7.8	8.1	5.6
Standby / Sleep (W)	∼1.1	1.0	1.3	?.?

RAID Background (1 / 3): Disk Mirroring

(a) Disk Mirroring



Advantage(s): Can operate w/a failure of a HDD
Can read in parallel

Disadvantage(s): Cost (2x the drives)
Power

RAID Background (2 / 3): Disk Striping

(b) Disk Striping – striping units are distributed across multiple disks.

Example(s):

4 drives (# 0, 1, 2, 3). A file of 5 blocks
Store block b on drive b % 4.

Advantage(s): Performance (parallel reads/writes)

Disadvantage(s): Increased prob. of disk system failure
No data replication

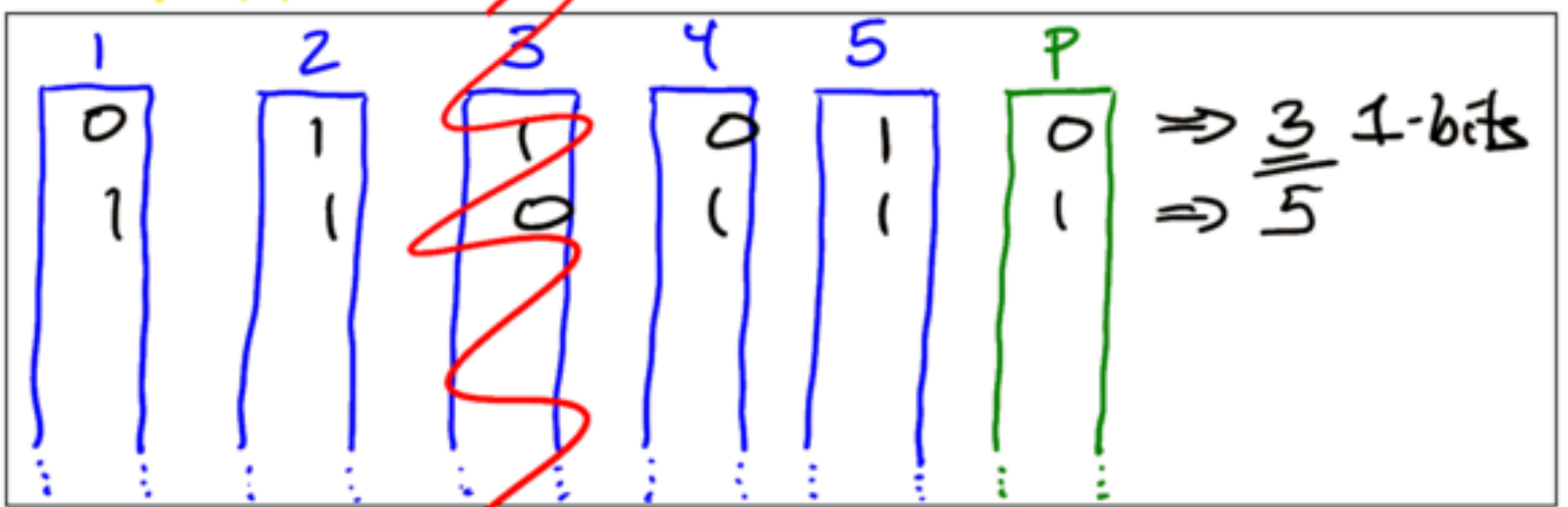
RAID Background (3 / 3): Parity Bits

(others: even, mark, none)

(c) Parity Schemes

odd parity (for this example)
# of 1-bits to be odd

Example(s):



Advantage(s): Can recover 1 failed drive

Disadvantage(s): Buy the parity drive (cost)
Parity drive is popular
only good for 1 failure

Detour: Independent Event Probabilities (1 / 3)

First, some set and probability review!

1. DeMorgan's Laws for Sets:

$$\overline{A \cup B} = \overline{A} \cap \overline{B}$$
$$\overline{A \cap B} = \overline{A} \cup \overline{B}$$

2. For a sample space S and an event $E \in S$,

   the probability of E's occurrence is:

$$p(E) = \frac{|E|}{|S|}$$

3. $\sum_{e \in S} p(e) = 1$

Detour: Independent Event Probabilities (2 / 3)

Next, Independent Events:

4. Events A and B are *independent* when . . .

$$P(A \cap B) = P(A) \cdot P(B) \qquad 1/2 \cdot 1/2 = \underline{1/4}$$

Ex: Flip a coin twice

$P(H \cap H) = \underline{1/4}$ $\qquad P(H \text{ on } 1^{st} \text{ flip}) = 1/2$

" " 2^{nd} " $= 1/2$

5. Recall: Principle of Inclusion/Exclusion for 2 Sets is:

$$|M \cup N| = |M| + |N| - |M \cap N|$$

Applied to probabilities:

$$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$$

Detour: Independent Event Probabilities (3 / 3)

Probabilities for Independent Events (cont.):

Recall:

4. $p(A \cap B) = p(A) \cdot p(B)$
5. $p(A \cup B) = p(A) + p(B) - p(A \cap B)$

6. Combining (4) and (5):

$$p(A \cup B) = p(A) + p(B) - p(A) \cdot p(B)$$

7. And thanks to DeMorgan's Laws and (4):

$$p(\overline{A \cup B}) = p(\overline{A} \cap \overline{B}) = p(\overline{A}) \cdot p(\overline{B})$$

Probability of Hard Disk Drive Failures (1 / 4)

Factors contributing to HDD failures:

Temperature, manufacturer, age, qty of scan errors, qty of sector remappings

How often does a 'young' (1-3 years old) HDD fail?

Annualized Failure Rate (AFR) = % of devices failing in a year

Refs: http://research.google.com/archive/disk_failures.pdf
https://www.backblaze.com/blog/~~best-hard-drive/~~ backblaze-drive-stats-for-q2-2022/

Announcements

- Program #1 is due now!
  - Can use late days if you wish
  - TAs will have grades for you before the due date of ...
- Handout: Program #2 - Linear Hashing Lite
  - I will provide a separate video to help clarify the expectations for the linear hashing index.
- Today: Finish RAID, jump ahead to hash-based indexing, discuss Program #2, then jump back.

Probability of Hard Disk Drive Failures (2 / 4)

What is the p_f for a striped 2-disk system?

$\Rightarrow$ Remember, the system fails when either drive fails!

(Let D#$_f$ be the event of Disk # failing.)

$$
\begin{aligned}
p_f &= p(\mathsf{D1}_f \cup \mathsf{D2}_f) && \text{Either or both!} \\
&= p(\mathsf{D1}_f) + p(\mathsf{D2}_f) - p(\mathsf{D1}_f) \cdot p(\mathsf{D2}_f) && \text{Princ. Inc./Ex. \&} \\
&= 0.02 + 0.02 - (0.02)^2 && \ldots \text{Indep. events} \\
&= 0.0396 \ (3.96\%)
\end{aligned}
$$

Probability of Hard Disk Drive Failures (3 / 4)

New point of view: Be an optimist!

The probability that Disk D# does not fail:

$$p(\mathsf{D\#}_{nf}) = 1 - p(\mathsf{D\#}_{f}) = 1 - 0.02 = 0.98$$

What is the p_{nf} for a striped 2-disk system?

$$\begin{aligned}
p_{nf} &= p(\overline{\mathsf{D1}_f \cup \mathsf{D2}_f}) && \text{[Neither fails!]} \\
&= p(\overline{\mathsf{D1}_f} \cap \overline{\mathsf{D2}_f}) && \text{[De Morgan's]} \\
&= p(\mathsf{D1}_{nf} \cap \mathsf{D2}_{nf}) && \text{[} \overline{\mathsf{D\#}_f} = \mathsf{D\#}_{nf} \text{]} \\
&= p(\mathsf{D1}_{nf}) \cdot p(\mathsf{D2}_{nf}) && \text{[Independent events assumed]} \\
&= p(\mathsf{D\#}_{nf})^2 && \text{[Foreshadowing ...]} \\
&= (0.98)^2 && \text{[From above]} \\
&= 0.9604 \text{ (96.04\%)} && \text{[} = 1 - 0.0396 \text{]}
\end{aligned}$$

Probability of Hard Disk Drive Failures (4 / 4)

What if we have *dozens* of HDDs? Say, three dozen?

No problem; being optimistic scales nicely!

$$
\begin{aligned}
p_{nf} &= p(\overline{\mathsf{D1}_f \cup \ldots \cup \mathsf{D36}_f}) && \text{[None fail!]} \\
&= p(\overline{\mathsf{D1}_f} \cap \ldots \cap \overline{\mathsf{D36}_f}) && \text{[Massive De Morgan's]} \\
&= p(\mathsf{D1}_{nf} \cap \ldots \cap \mathsf{D36}_{nf}) && [\ \overline{\mathsf{D\#}_f} = \mathsf{D\#}_{nf}\ ] \\
&= p(\mathsf{D1}_{nf}) \cdot \ldots \cdot p(\mathsf{D36}_{nf}) && \text{[Independent events assumed]} \\
&= (0.98)^{36} && \text{[From last slide]} \\
&= 0.4832\ldots \text{ (48.32\%)} && [\ p_f = 1 - p_{nf} = 0.5168\ ]
\end{aligned}
$$

Remember: Assuming independence is convenient, not realistic!

RAID: Redundant Arrays of Independent* Disks (1 / 2)

* *Originally "Inexpensive"*

Level 0: Striped Volume (N data disks)

⇒ Striping only!

- Good write performance
- Cannot recover from a drive failure
  (Even if we're making backups.)

Level 1: Mirrored (N data disks + N mirror disks)

⇒ Mirroring only!

- Opportunity for parallel reads
- Cost: 2x the # of drives!

RAID: Redundant Arrays of Independent Disks (2 / 2)

Level 5: Block–Interleaved Distributed Parity (N+1 disks)

- striping unit is 1 block
- parity blocks are distributed across disks
- controller design is kinda complex

Level 6: "Double Parity" (N+2 disks)

- Level 5 w/ 2 parity schemes
- Can recover from 2 concurrent failures
- same read perf. as Level 5, but slower write perf.

SSDs: Solid–State Device (Flash) Storage

- NAND–based non–volatile RAM NVRAM
- Not a new idea: Used to have "RAM drives"

  (Even though the 1981 IBM PC had 256 KB RAM – max!)

Advantage(s): No mechanical delays or failures
Excellent read perf.
Consumes little physical space

Disadvantage(s): Write speed lags (erase then write)
# erasures ≈ 100K – 1000K
more expensive than HDDs

Review of Internal Hashing

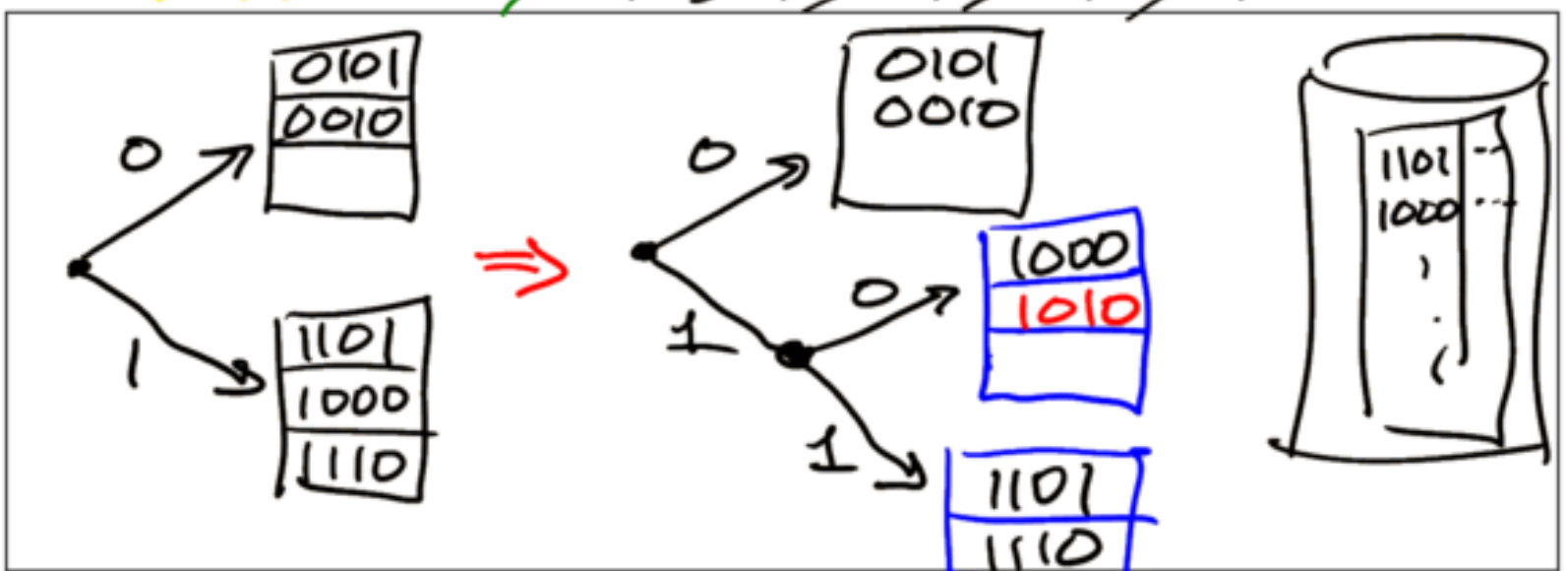
- Goal: $O(1)$ search performance
- Key $\rightarrow$ Hash Coding $\rightarrow$ Compression Mapping $\rightarrow$ Hash Table Index
- Collision Resolution: Chaining v. Open Addressing
- Problem: How can we do hashing on secondary storage?

Dynamic Hashing

Two components:

- A tree-structured directory
- Leaves are disk blocks holding the index entries.

Example(s): Insert 1101, 1000, 0101, 0010, 1110, and 1010:



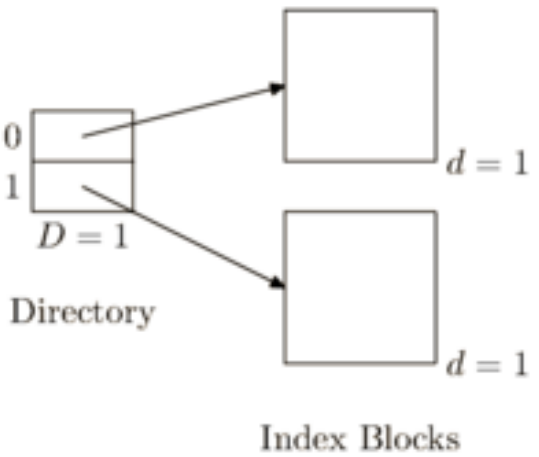
3 entries / block

Problem: How do I store the directory on disk?

Extendible Hashing: Basics

Improvement over Dynamic Hashing:

Directory is an array ⇒



- Each index block has a local depth (d), $d \geq 1$
- The directory has a global depth (D), $D \geq \max(d_i)$
- Directory has k^D pointers, where k is the cardinality of the alphabet set.
  - Σ_6: When keys are binary, $k=2$

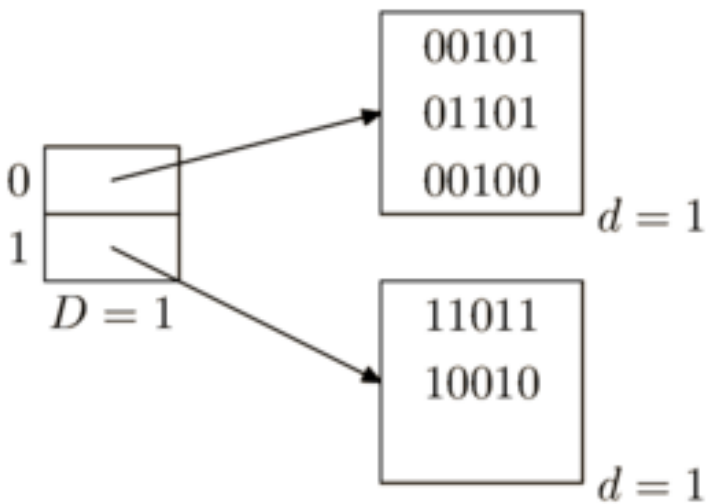
Extendible Hashing: Insertion (1 / 2)

When a key is inserted into a full index block:

- The block becomes k blocks
- The depth of each is one more than the original's
- Existing content is distributed to the new blocks
- If any $d > D$, split ('double') the directory:
  - increase global depth by one
  - create new directory of k^D pointers
  - copy existing block pointers
  - add pointers to new blocks

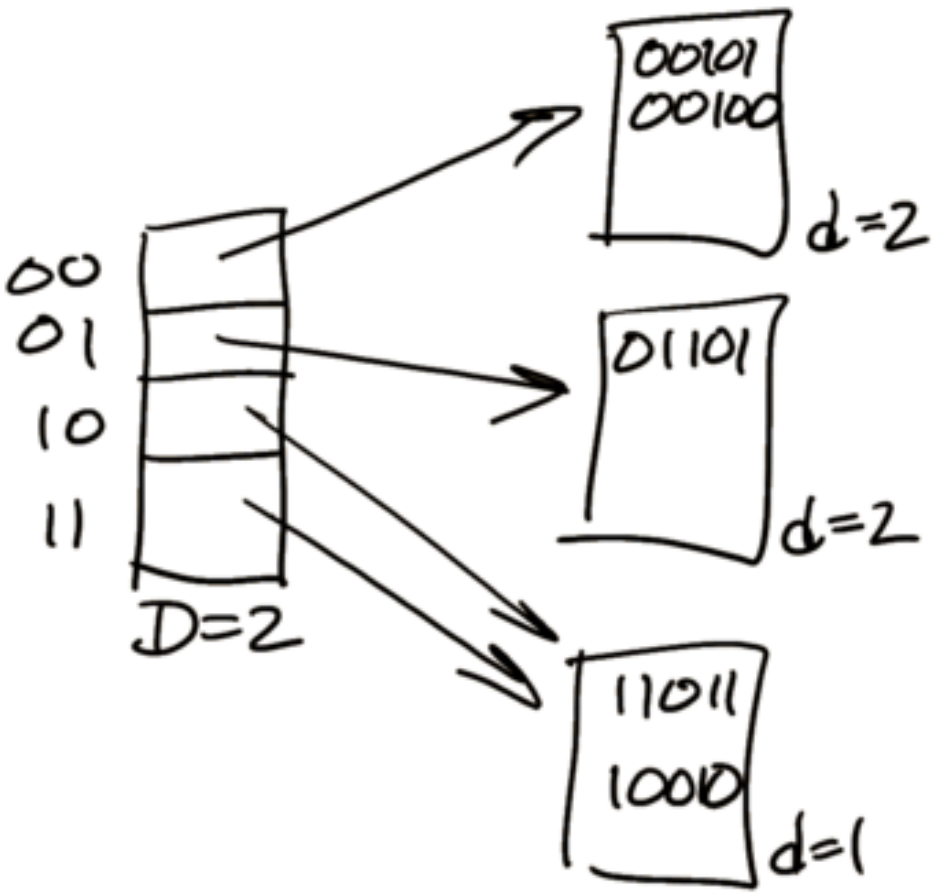
Extendible Hashing: Insertion (2 / 2)

After Inserting 11011, 00101, 01101, 10010, and 00100:



(Assume max. 3 keys/node)

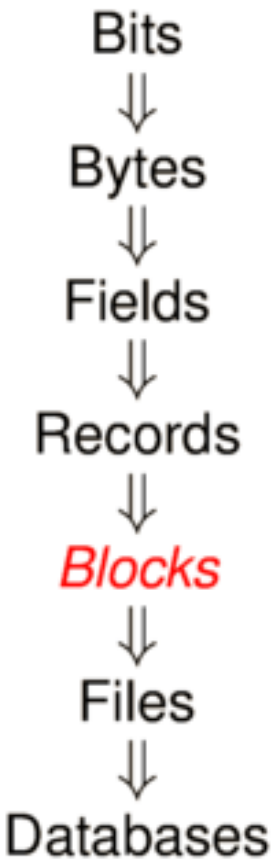
After Inserting 01110:



Announcements

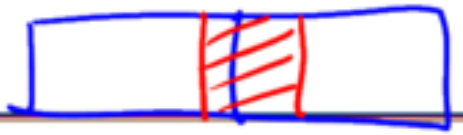
- Program #2 is due a week from Wednesday.
  - The video that shows how to get started is available in D2L.
  - Any questions?
- Today:
  - Finish last slide on Extendible Hashing
  - Jump back to key indexing ideas
  - (Maybe) Start B-Trees
- CS Career Fairs are this week!

File Granularity Hierarchy



File Blocking (1 / 2)

1000
300



Definition: Blocking Factor (bf)

The # of records that can be stored in a block

$$bf = \left\lfloor \frac{\text{block size}}{\text{record size}} \right\rfloor$$

(assuming records are not split between blocks)

Definition: Internal Fragmentation

Unallocatable storage within a block.

8192

Example(s): Block size = 8K bytes; Record size = 55 bytes.

$bf = \left\lfloor \frac{8192}{55} \right\rfloor = 148$ recs/block

$8192 - 148 * 55 = 52$ bytes of internal frag.

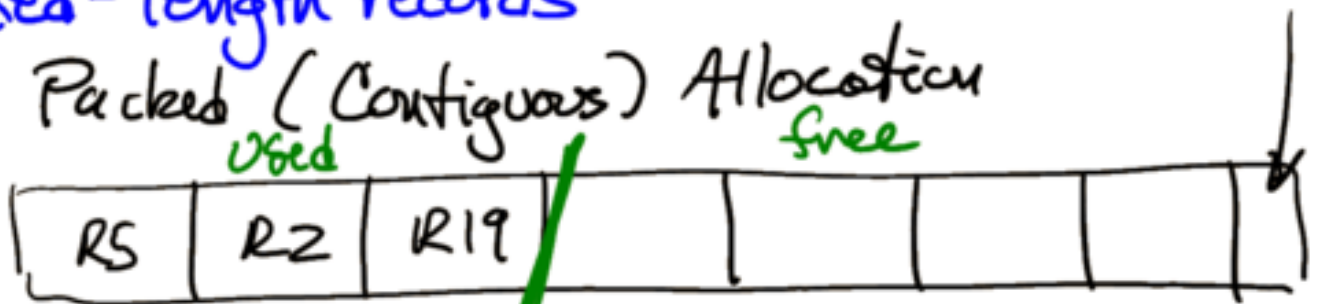
File Blocking (2 / 2)

count or bitmap

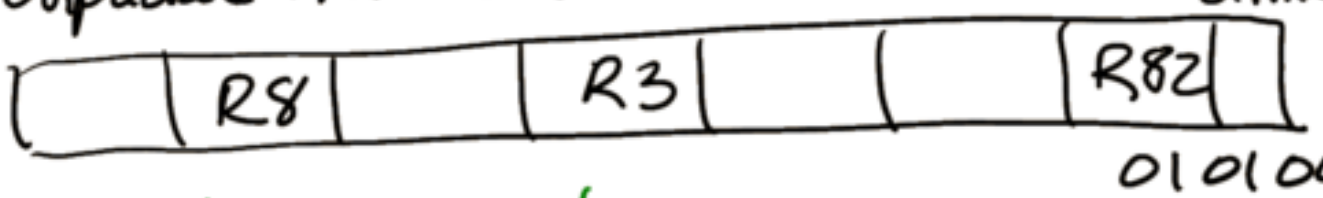
Locating records within blocks:

(1) Fixed-length records

(a) Packed (Contiguous) Allocation

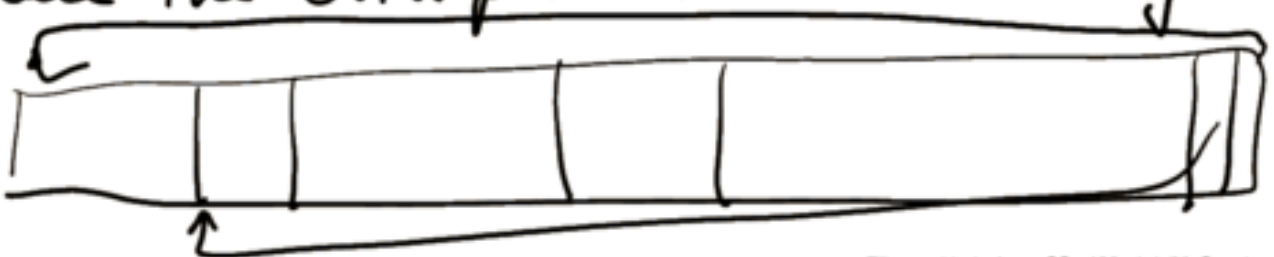


(b) Unpacked Allocation



(2) Variable-length records

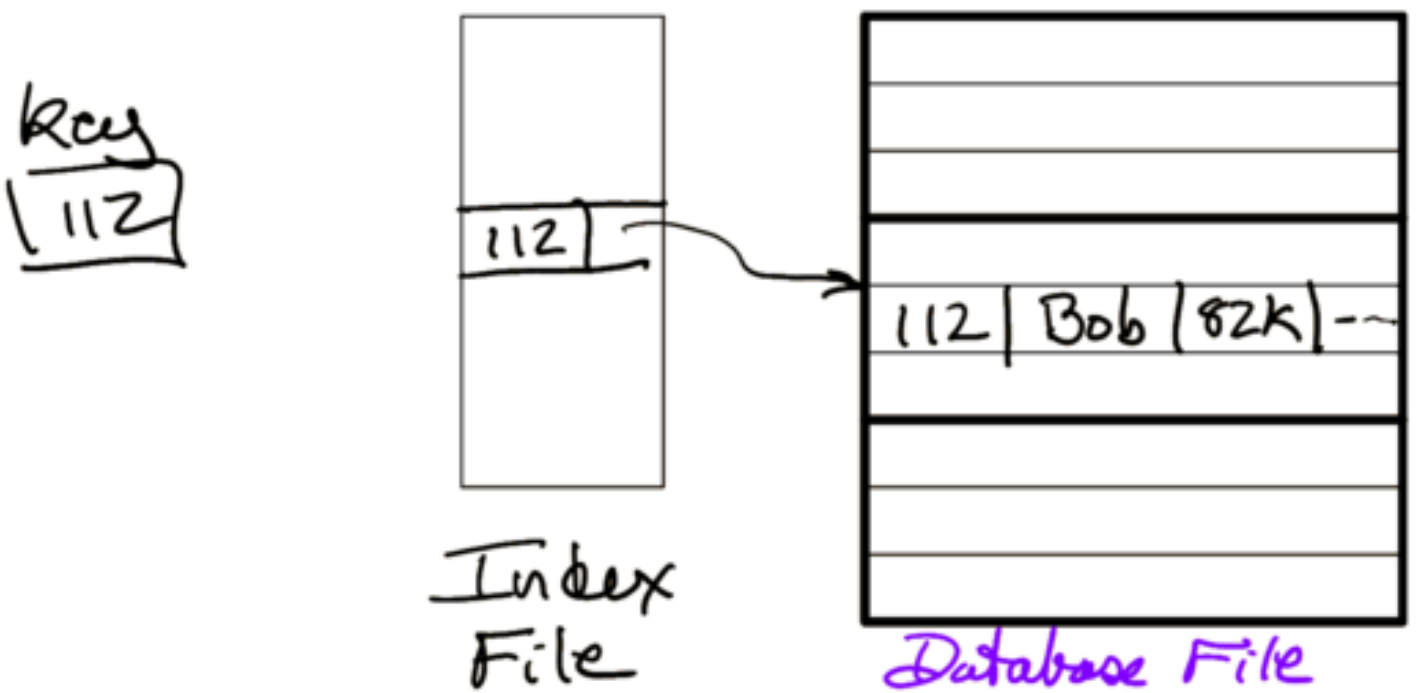
Replace the bitmap with a slot directory



Indexing

Definition: Index

A file containing structured references to records of another file.



A Few Words about Keys

Some of the types of keys:

- Candidate Key – A key able to uniquely identify a record!
  Ex: StdID, NetID, ~~SSN~~
- Primary Key – The CK chosen to be the identifier
- Secondary Key – Any non-CK
  Ex: SSN, Name, birthday, ---

- Sort Key – The key used to order the records of a file.
- Search Key ---

One Classification of Indices

(1) Ordered

(a) Single-level

Ex: Sorted file of index records

(b) Multi-level

Ex: B^+-Tree

(2) Unordered Index

- Typically hash-based

Ex: Extendible Hashing

Primary Index (1 / 2)

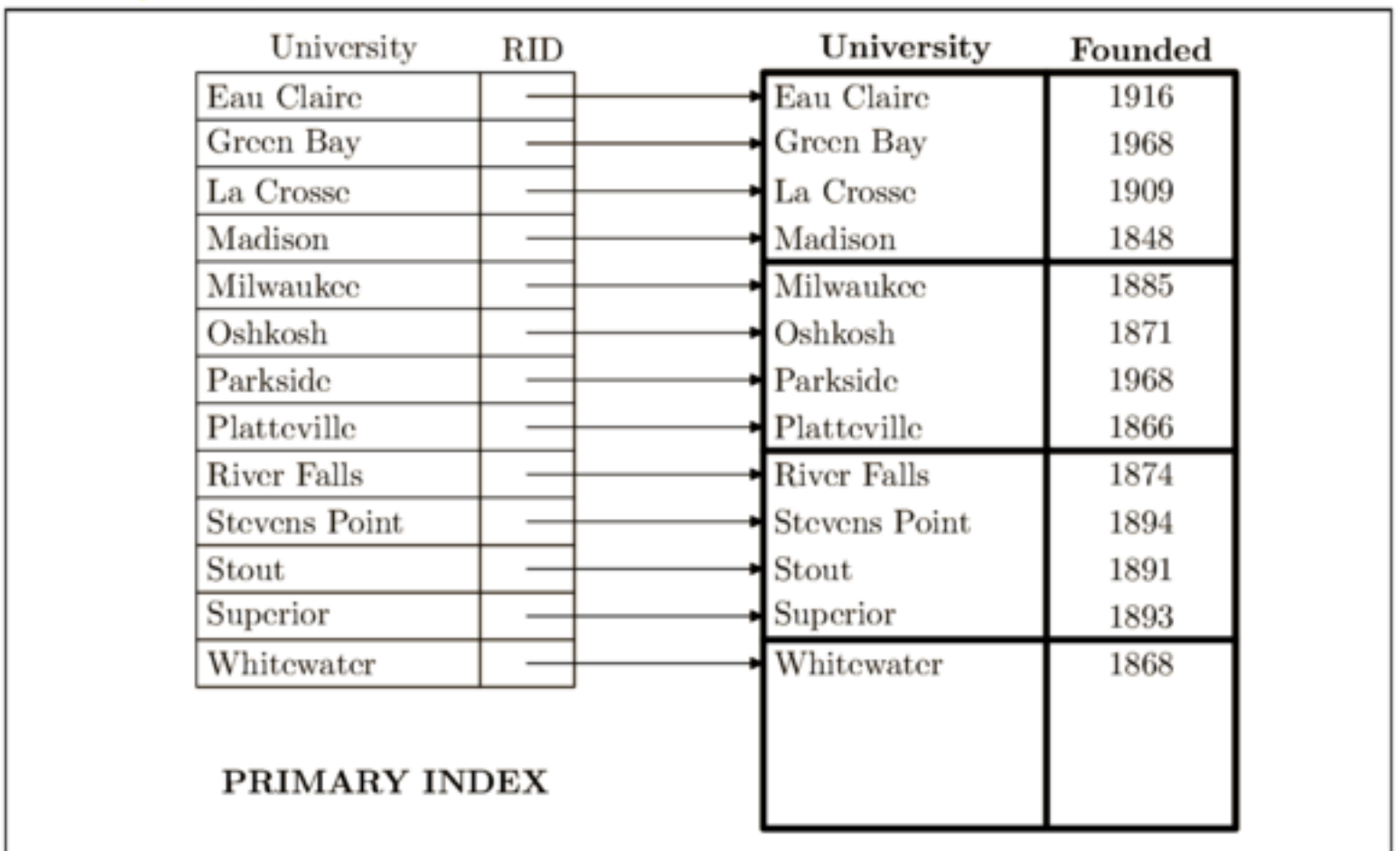
Characteristics:

- The indexed field is a candidate key.
- The index records are sorted on the key.
- The DB file records are sorted on the key.

Notes
- Can have at most one per file
- The term "primary index" has many definitions.

Primary Index (2 / 2)

Example(s):



Clustered Index (1 / 2)

Characteristics:

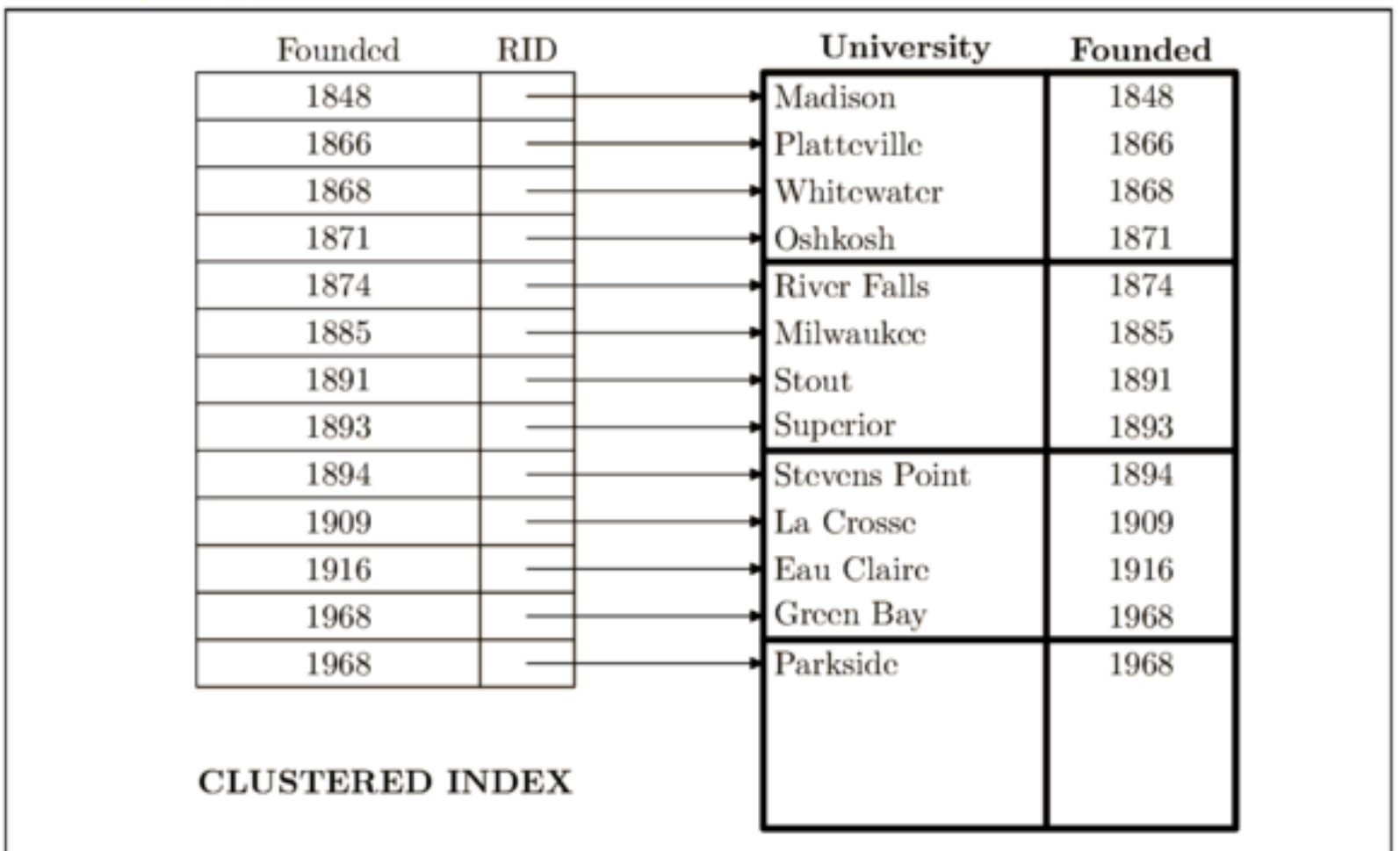
Not a CK!

- The indexed field is a secondary key.
- The index records are sorted on the key.
- The DB file records are sorted on the key.

Notes:
- per DB file, the # of primary indices plus the # of clustered indices can be no greater than one.
- Some see primary indices as a special case of clustered.

Clustered Index (2 / 2)

Example(s):



Secondary Index (1 / 2)

Characteristics:

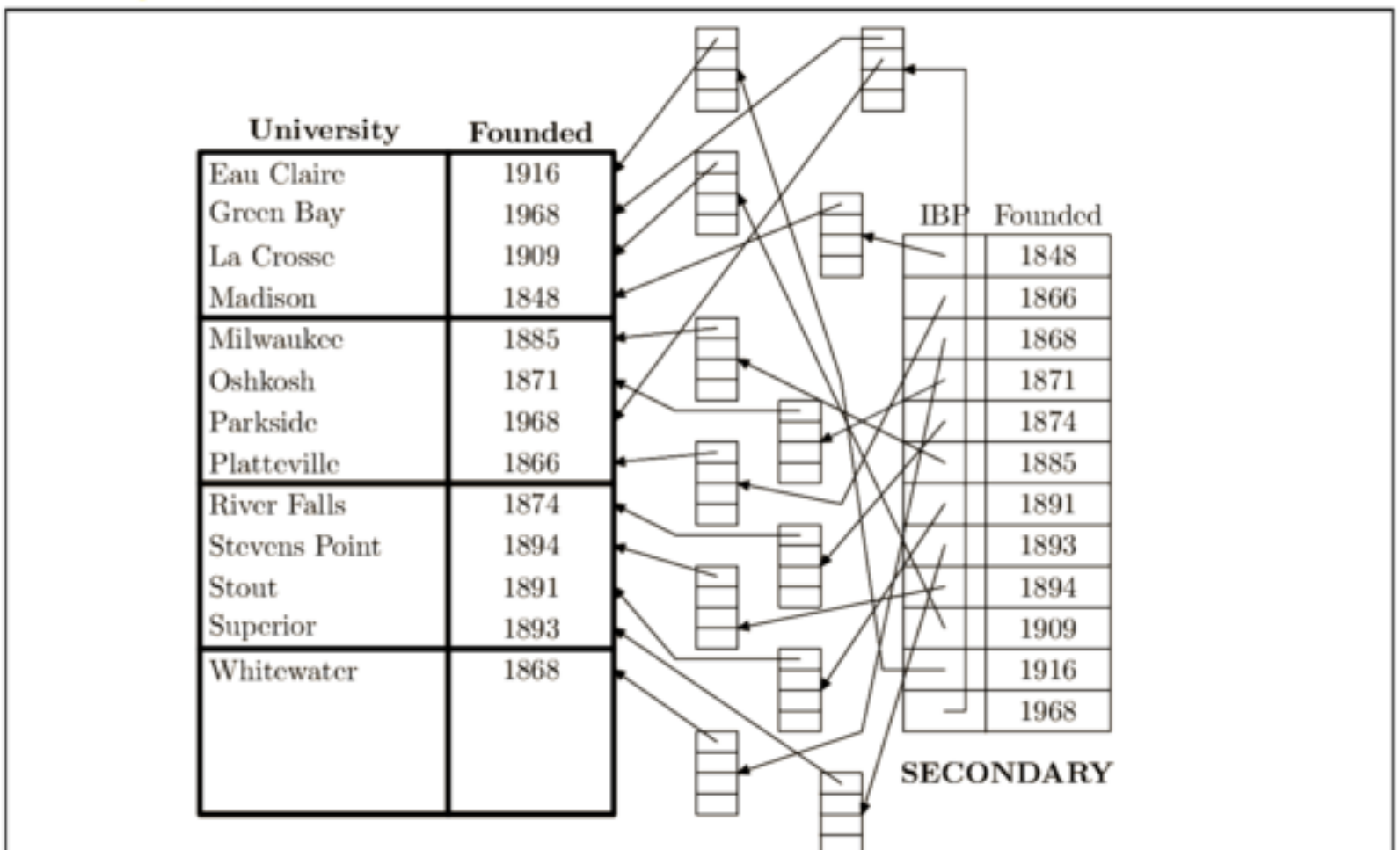
- The indexed field is any field! .
- The index records are sorted on the key.
- The DB file records are not sorted on the key.

Notes

- Can have as many of these as you want
- special index construction is needed.

Secondary Index (2 / 2)

Example(s):



Extendible Hashing: Deletion

Question: Do you have lots of disk space available?

If so: Just delete the entry
(leave the space for future insertions)

If not: Collapse the sibling nodes
and shrink the directory if
necessary/possible

Announcements

- Program #2 (Linear Hashing Lite) is due in a week (Wed, Sept. 21)
  - you'll need your .bin file that Program 1 (Part A) created
- Next assignment will be Homework #1, to be assigned next Wed., and due a week later
- Exam #1 is in 3 weeks.

Another Index Categorization: Dense vs. Sparse (1 / 2)

Dense Indices:

- Hold one index record for every file record (excepting perhaps dup. keys)
- Permits existence queries using just the index

Sparse Indices:

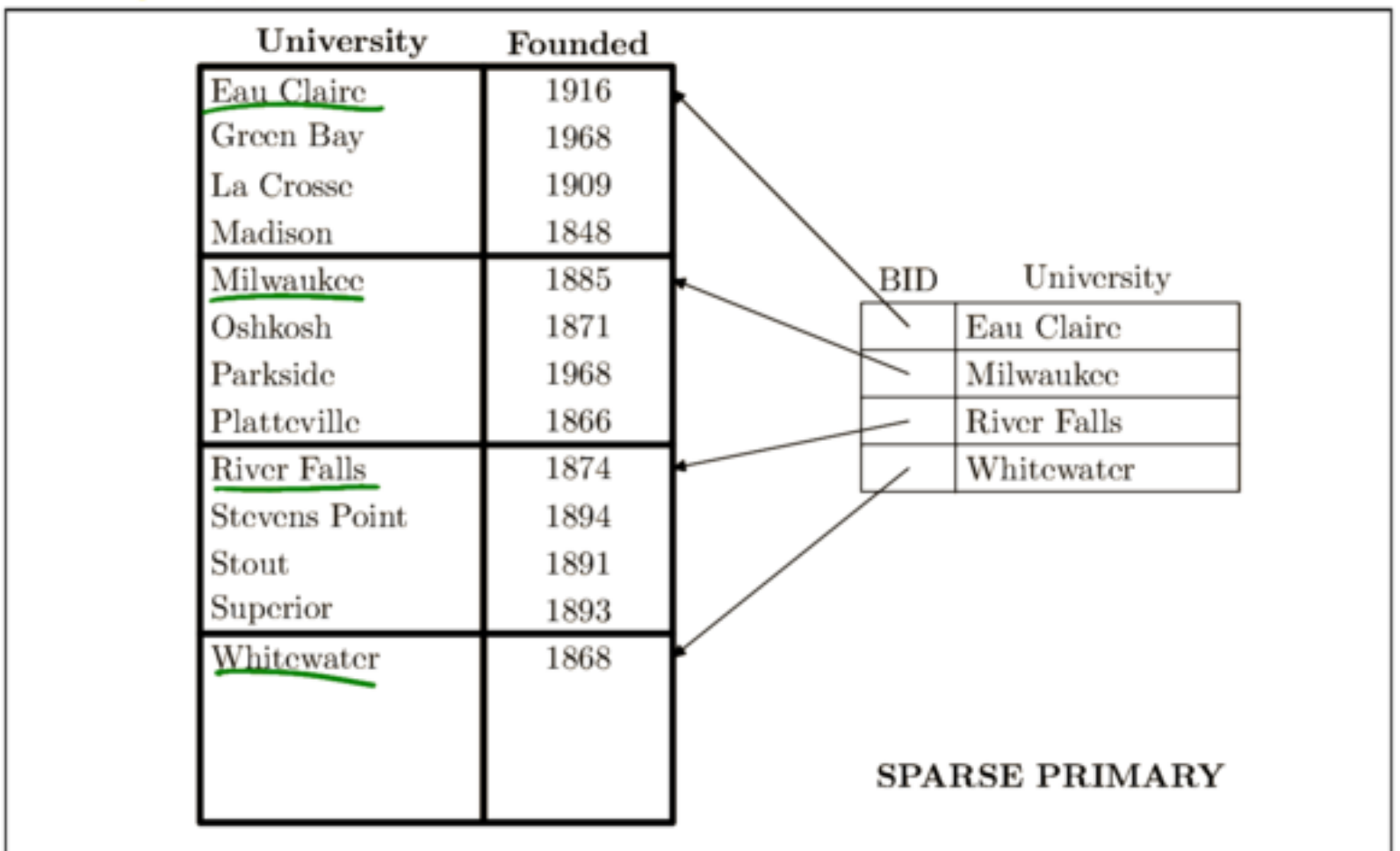
- Index has a subset of the field values of the key
- Smaller, ∴ faster to search

Notes:

- All of the examples of ordered indices from Monday are dense.
- Any ordered index type can be either dense or sparse.

Another Index Categorization: Dense vs. Sparse (2 / 2)

Example(s):

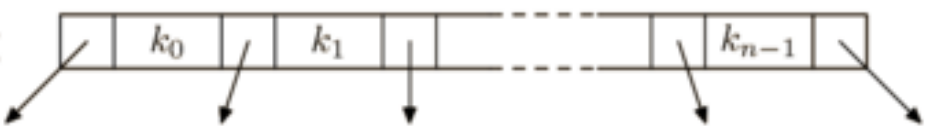


B–Trees: Structure

But first: Know that "B" does not stand for "binary"!

"Bayer"? (Rudolf Bayer & Edward McCreight, '72)
"Balanced"? (It is!) "Boeing"? (McCreight's employer?)

Structure of a B–Tree node:



— A node holding n keys holds $n + 1$ pointers

— Each key is stored in the index exactly once (∴ dense)

— A node's keys are stored in (ascending) sorted order

— Pointer 0's subtree has all keys $<$ key k_0

— Pointer i's subtree has all keys $> k_{i-1}$ and $< k_i$

— Pointer n's subtree has all keys $> k_{n-1}$

B-Trees: Definition

Definition: B-Tree of Order M (*a la* D. Comer[♮])

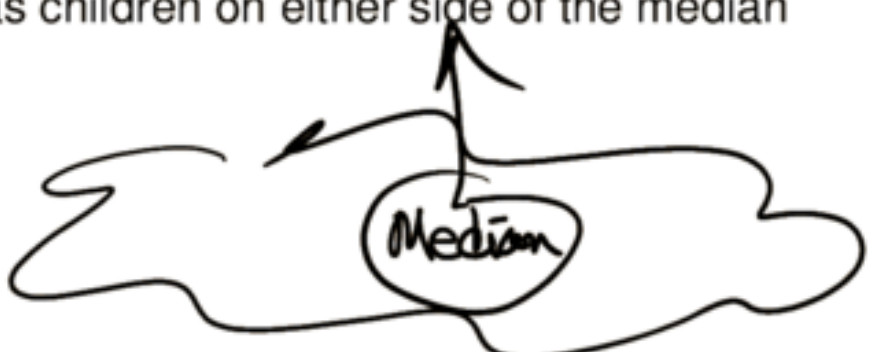
- Each node contains at most 2M keys (∴ 2M+1 pointers)
- Each node (except the root) must contain at least M keys.
- A non-leaf node node has at least 2 children
- All leaf nodes are at the same level
- A non-leaf node with n keys has exactly n+1 children.

[♮] Comer, D. "The Ubiquitous B–Tree," ACM Computing Surveys 11(2), June 1979, pp. 121-137.

B-Tree: Insertion (1 / 2)

- Find the leaf node that should contain the new key value
- If leaf has capacity, insert the key into it.

  Otherwise: (when leaf node has 2M keys)
  - Form a set of the leaf's keys plus the insertion key
  - Promote the set's median value to the parent
  - Create two nodes to hold the key values that are $<$ and $>$ the median, respectively.
  - Attach nodes as children on either side of the median

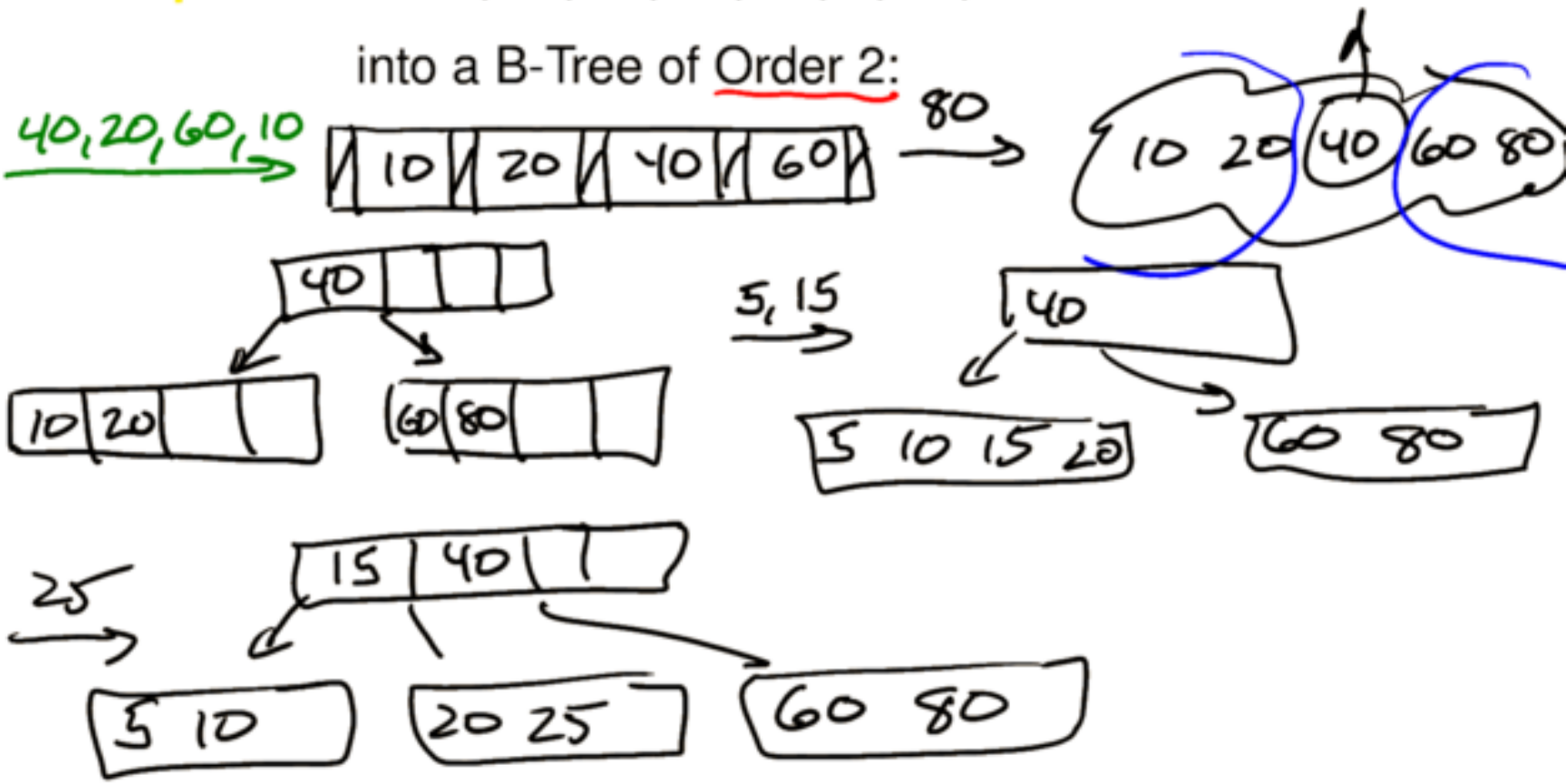


2M+1 keys

B-Tree: Insertion (2 / 2)

Example: Insert 40, 20, 60, 10, 80, 5, 15, and 25

into a B-Tree of Order 2:



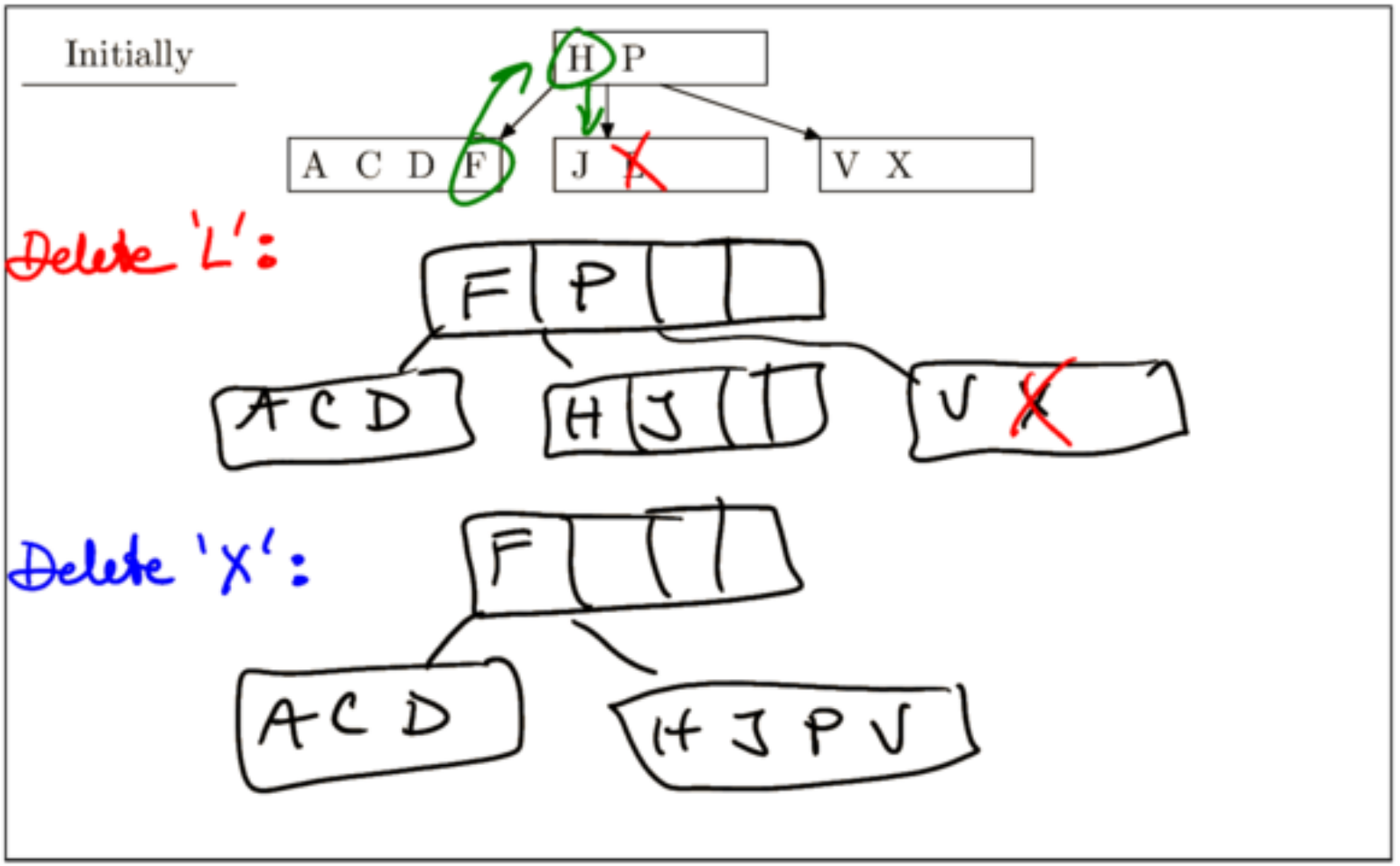
B-Tree: Deletion (1 / 2)

When a deletion leaves a node under-full:

- If the under-full node is a leaf node:
  - If a neighboring sibling has above-minimum occupancy, borrow:
    - Move separating value from parent to under-full node
    - Move appropriate value (smallest / largest) from neighbor to parent
  - Otherwise, concatenate:
    - Merge node, a neighboring sibling, and the parent's separating value into one node
    - (Note that this can leave the parent under-full, so recurse!)
- Otherwise, the under-full node is an internal node:
  - Replace deleted key with its inorder predecessor or successor
  - Recurse if necessary

B-Tree: Deletion (2 / 2)

Example(s): Still assuming M = 2



B-Tree: Capacity

What is the key capacity of a B-Tree of Order M?

Example(s): Order 128

First Level (the Root): 256 keys

Second Level: 257 nodes * 256 keys
= 65,792 keys

Third Level: 257^2 nodes * 256
= 16,908,544 keys

Total = 16,974,592 keys.

B-Tree: Order Determination

The Idea: Select order to best fit disk block capacity

Remember: A node of a B-Tree of Order M can hold

$2M$ keys and $2M + 1$ pointers

Example(s): Assume 8K bytes/block, keys of 12 bytes, ptrs of 8 bytes

8192

The amount of storage required by a node, in terms of M:

$$(2M)(12) + (2M+1)(8) \leq 8192$$

$$M \leq 204.6$$

So $M = 204$

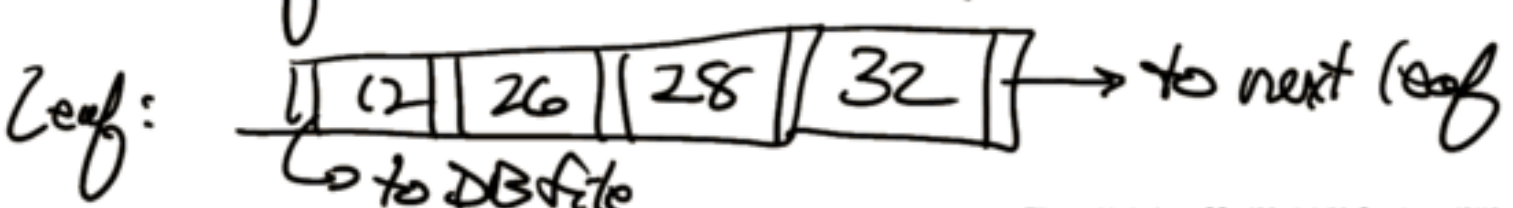
Announcements

- Program #2 is due Wednesday @ the start of class
  - Submit your code via turnin to the cs460p2 folder
- Up next: Homework #1
  - Assigned Wednesday, due next Wednesday
- Program #1 was graded & scores emailed to you over the weekend.
  - Have grading concerns? Start by directing your questions to Priya & Aayush. If you reach an impass, come talk with me about it.

B^+–Tree: A B-Tree for Indexing

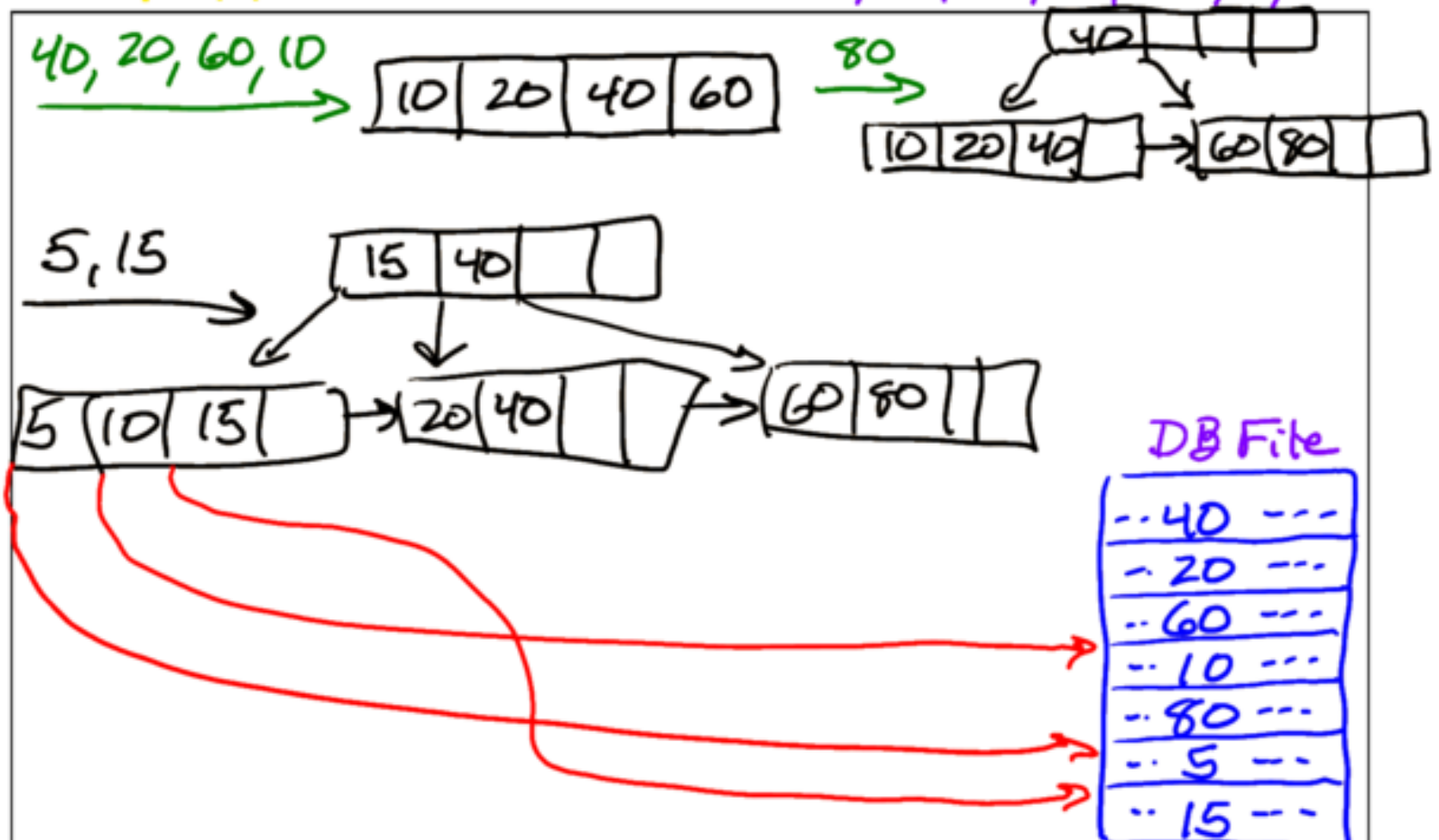
Like a B-Tree, but:

- All of the keys are stored in leaf nodes
- Copies of some keys occupy internal nodes (to maintain the tree structure)
- Leaf nodes are linked sequentially (L→R) to form the sequence set
  - facilitates linear searching
  - requires no additional pointers



B$^+$–Tree: Insertion

Example(s): Order 2. Insert 40, 20, 60, 10, 80, 5, and 15.



B^+–Tree: Advantages and Disadvantages over B-Trees

Advantage(s):

- Has built-in disk pointer space in leaf nodes
- Supports exact-match ("find 26") and range ("Find 20-60") queries.

Disadvantage(s):

- "waste" of storage in internal nodes (but a tiny amount)
- insertions / deletions are a little more complex.

Topic 4:

E-R

DB Design and the Entity–Relationship Model

Review of File Schemata

- Recall: Fields $\Rightarrow$ Records ($\Rightarrow$ Blocks) $\Rightarrow$ Files
- A record represents a real–world item or concept

  Example: A student record in a grading program
- A basic DB file's records all have the same construction

  (Same fields, same types, same order)
- Identification:

  — Fields: By assigned name

  — Records: By primary key
- Together, these items define the file's schema

Date's Supplier-Part-Project Schema

Also see the SPJ Schema handout!

Used by C. J. Date in his papers and textbooks.

Consists of four files:

Supplier (S)	S#	Sname	Status	City	
Part (P)	P#	Pname	Color	Weight	City
Project (J)	J#	Jname	City		
SPJ	S#	P#	J#	Qty	

Notes:

- We underline the PKs
- All fields of a composite key are underlined.

Three Tangents

- These are topics of importance to the creation of file schemas.
- They need to be introduced sometime; might as well be now!
- They are:
  - Nulls
  - Foreign Keys
  - A Few Types of Data Integrity

Nulls

Definition: Null

A null is a marker that indicates that a field does not have a value.

Notes:

- The phrase "null value" is contradictory here.
- Nulls are useful when values are not known.
  Ex: Students w/o a local address
- Nulls are somewhat controversial
  - Relational Model is based on logic
  - But they are useful!

Foreign Keys

Definition: Foreign Key

A field in one file whose values are drawn from the PK field of (usually another) file.

Notes:

- Unlike PKs, FKs may be null.
- Every FK value must also be a PK value
- Ex: In SPJ schema, the SPJ table holds 3 FK fields.

A Few Types of Data Integrity

Foreign keys are essential to three types of data integrity:

1. Key Constraint
   - PKs and FKs are designed into a DB.

2. Entity Integrity
   - No PK field may be null (b/c PKs are record IDs)

3. Referential Integrity
   - This is the name for the idea that an FK value must refer to an existing PK value

⇒ All take effort to enforce.

DB Design: Overview

- Very similar to software development processes
- Everyone has his/her own n step design process
  - The one we'll present is rather generic
- Some ideas to keep in mind:
  - Any design process is iterative
  - Processes can be categorized as being either . . .
    - top-down vs. bottom-up, or
    - data-driven vs. function-driven, or
    - . . .

DB Design: Phases 1 & 2

Phase 1: Requirements Analysis – Learn what the customer needs!

... by asking lots of questions!

Phase 2: Conceptual Design – create the conceptual schema

- the high-level description of the DB's structure
- includes data requirements, data relationships, data constraints,
- does <u>not</u> include implementation details
- example modeling tool: The E-R Model

DB Design: Phases 3 & 4

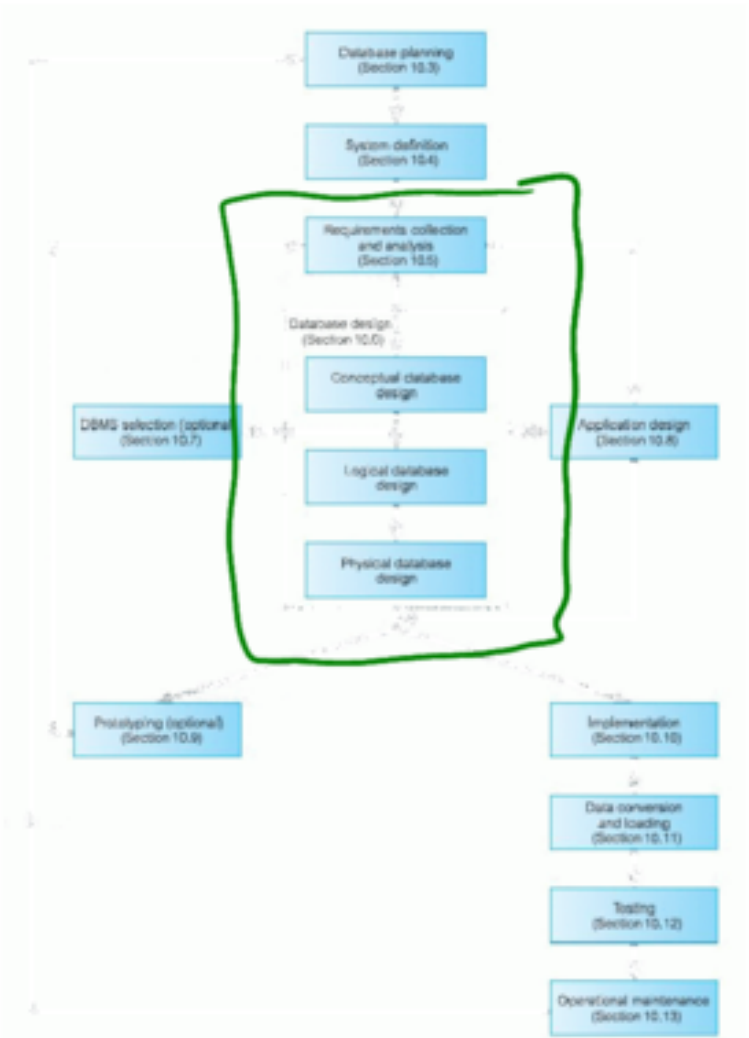
Phase 3: Logical Design – convert conceptual design into a DBMS' implementation data model (See Topic 5)

- Ex: The Relational Model
- Conversion includes data normalization

Phase 4: Physical Design – specify the DB's storage requirements

- Such as: indices, record order, data types
- all based on expected usage of the DB. (Learned in Phase 1.)

DB Design: In Context



Credit:

Connolly/Begg, 6/e,

Figure 10.1, p. 300.

Announcements

- Program #2 is due now!
  - Submit files using turnin to cs460p2 folder.
- Handout: Homework #1 ← Note: #9's (13.7's) topic will be covered Monday.
  - Pay heed to the directions:
    - Submissions must be PDFs (yes, you can write answers by hand & scan to PDF)
    - Upload your PDF to Gradescope.
  - you can use at most 1 late day on homeworks because ...
  - we will be making solutions available 24 hours after the due date/time, to help you study for ...
- Exam #1 is two weeks from today (Oct. 5th).

What are 'Entities'?

Definition: Entity

An entity is an instance of a general classification

DBs store representations of entities

Example(s):

Each of you is an instances of the class 'human'

Lincoln's Gettysberg Address is a speech instance.

What are 'Relationships'?

Credit: "Mother Goose and Grimm" by Mike Peters, 2009– 02–12

Definition: Relationship

A ↗ is an association between sets of entities.

Example(s):

Instructors teach classes.
Companies visit career fairs

One-to-One Relationships

Definition: One-to-One Relationship

In a relationship between two entity sets A and B, an entity from A is associated with at most one entity from B, and an entity from B is associated with at most one entity from A.

Symmetric

Example(s):

Employee manages Department

Patient — Vaccine Syringe

One-to-Many Relationships

Not Symmetrical!

(a.k.a. Many-to-One Relationships)

Definition: One-to-Many Relationship

In a →, one entity from A is associated with 0 or more entities from B, but each from B is associated with at most one from A.

Example(s):

Professor – Department (if multiple appts are not possible)
Barber – Client
(Biological) Father – Child

Many-to-Many Relationships

Definition: Many-to-Many Relationship (M-to-N)

A is a pair of 1-to-N relationships.

(Back to symmetry)

Example(s):

Employee – Project

Patient – Doctor

Other Varieties of Relationships

This list is by no means exhaustive! Some others:

- 1:1, 1:N, M:N with Varied Multiplicities

  Ex: Zero or One vs exactly one

- Ternary (a.k.a. 3-Way, Degree 3)

  Ex: SPJ Schema

  Ex: Course – Time – Room

- Recursive (a.k.a. Cyclic)

  Ex: Supervise : Employee – Employee

  – there are more! –

The E-R Model: Origins

- First proposed by Pin-Shan (Peter) Chen in a 1976 paper
  - — Extended many times since

    Example: Enhanced E-R (E-E-R) Model
  - — Has an annual conference devoted to it

    (Int'l Conf. on Conceptual Modeling)
- Easily the most popular conceptual model in use today
- Many of its ideas appear in the Unified Modeling Language (UML)
- Diagrammatic variants abound

An E-R Example (1 / 6): A Bank Database

Description:

Consider a (very) simple database for a bank. We need to store information about the bank's customers. Of course, the customers have accounts with the bank, and they perform transactions on those accounts.

Question: What are the entity sets for our database?

Customer
Account
Transaction

Hint: Use singular names!
Bank as an entity set? (NO)

An E-R Example (2 / 6): Fields

Question: **Which pieces of information do we need for each entity set?**

Customer: Name, SSN, CID, Address, ...

Account: Acct#, Balance

Transaction: Date, TID, Type, Amount, ...

Notes:

- A weak entity set is one whose existence depends upon another entity set. Ex: Transaction
  - symbol: Dashed underline is a partial key
- What Customer & Account are strong entity sets (have good primary keys)

An E-R Example (3 / 6): Relationships

Question: **Which relationships connect these entity sets?**

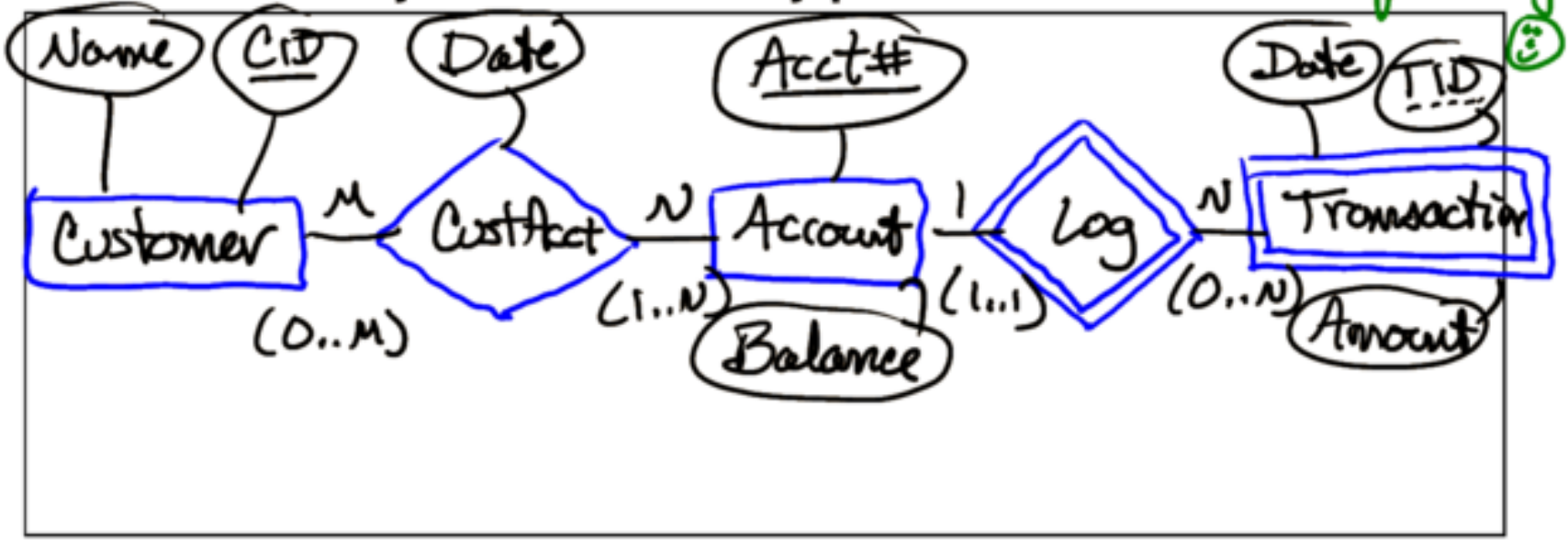
(1) Customer - Account (Cust Acct)
Cardinality: M:N
Attribute: Date of Creation

(2) Account – Transaction (Log)
Cardinality: 1:M

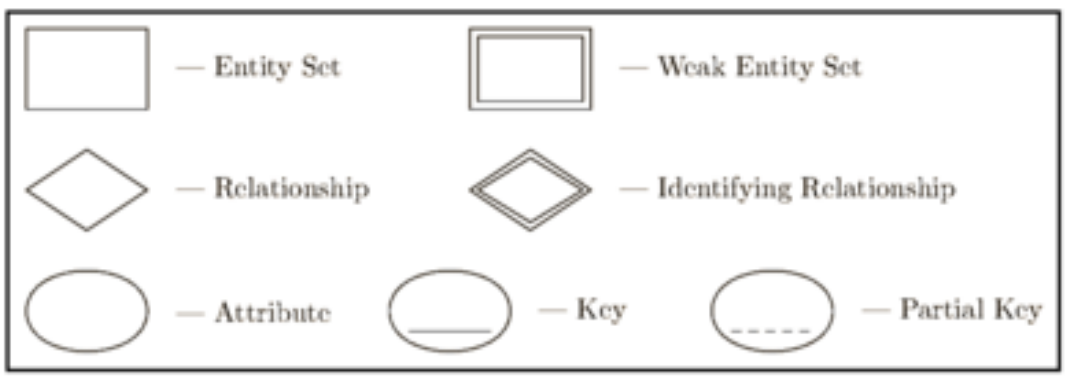
⇒ The relationship supporting a weak entity set is called an identifying relationship.

An E-R Example (4 / 6): Diagram (Chen's Notation)

Question: **Can you draw a lovely picture of all of this?** Depends; define "lovely"



Legend:

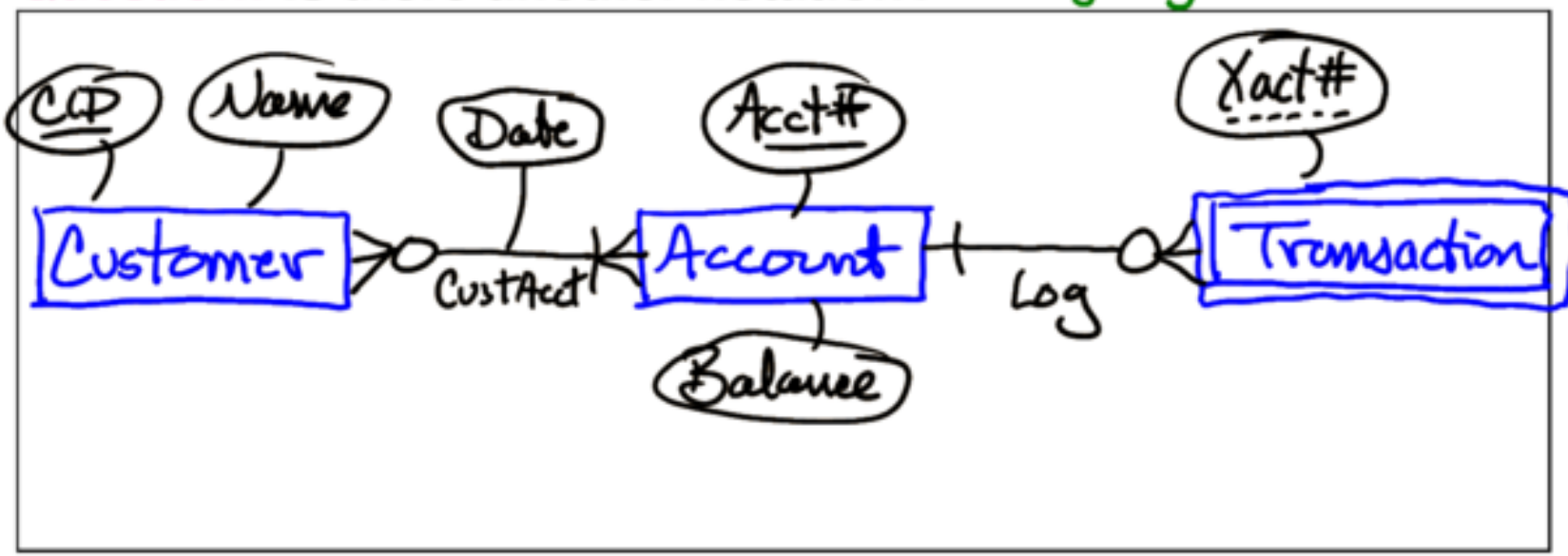


Announcements

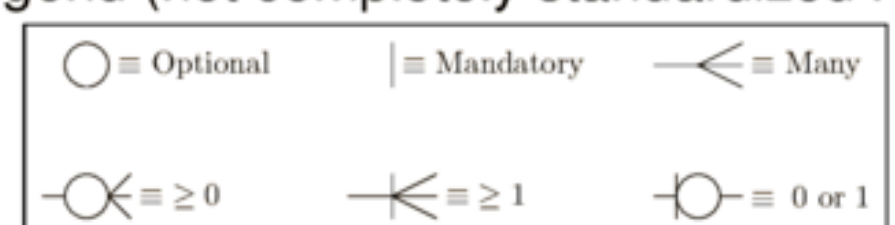
- Homework #1 is due @ the start of class on Wednesday.
  - Remember, submit your highly-legible PDF to Gradescope!
  - You can use a late day, but only one.
  - Answers will be posted after 3:30pm Thursday
- Exam #1 is Wednesday, Oct 5th.
  - Topics: 1-5 (hopefully!)
  - Sorry, no sample exam & no review session. (We're confident that you know how to study for exams by now!)
- We are behind pace, which isn't unusual, but is a problem...

An E-R Example (5 / 6): Diagram (Crow's Feet Notation)

Question: **Is there another notation?** Sure; why not?

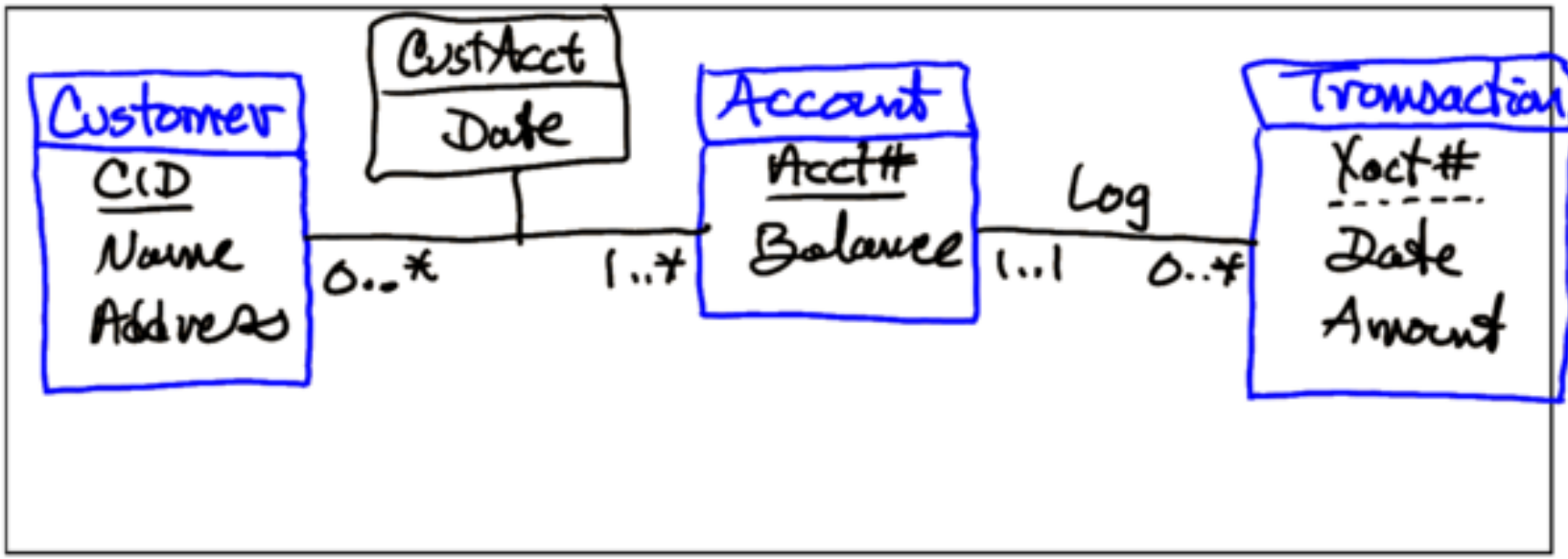


Legend (not completely standardized . . .):



An E-R Example (6 / 6): Diagram (UML Notation)

Question: **Doesn't UML include these concepts, too?** Yup!



Notes People mix and match from these styles
Lots of other notation options on each of these.

Another E-R Example: Faculty

Description:

University faculty members teach classes that are offered by departments. Faculty are members of departments. Each department has a chairperson.

Question: What are the entity sets and relationships?

E-R Modeling Rules of Thumb

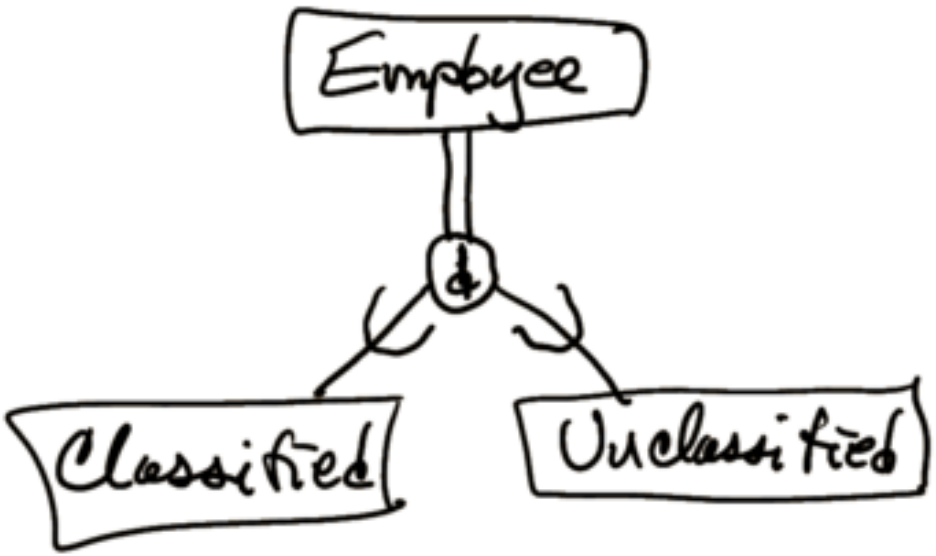
- Choose singular (v. plural) names for entity sets
- Naming relationships can be a challenge; concatentation of the names of participating entity sets is an option
- If you can't find a candidate key, perhaps the entity set is weak
  - If so, remember that the relationship is *identifying*.
- Mixing & matching notation is common
- Make your model as informative as possible

Enhanced E-R Model: Motivation

- The basic E-R Model was designed for 'business data'
  - Basically, text and numeric fields
- Now that computers are more common, more capable, and used for a wider variety of purposes, additional representational power is required to model user concepts.
- Generally, this is called *semantic modeling*.
- Some semantic modeling suggestions have been added to E-R modeling

EER: Specialization / Generalization ("is-a")

Consider inheritance in an O-O programming language . . .



Manager ⊂ Employee

═══ means "total specialization" ——— means "partial ..."

d means 'disjoint', o means 'overlap is ok', u means 'union'

EER: Aggregation ("has-a")

Several types:

1. An entity is formed from a collection of attribute values

   Ex: A person "has-a" name, id#, . . .

2. An entity formed from other entities

   Ex: A car is engine, tires, doors, . . .

3. An entity formed from a relationship to a relationship

   Ex: A job interview (relationship between Company and Applicant) resulting in a job offer

   Notation is often just a line between relationship diamonds.

Topic 5:

Implementation Data Models

A Wee Bit O' History

There are four data models of note:

1. Hierarchical (early 1960s, IBM)
2. Network (early 1970s, CODASYL/DBTG)
3. Relational (early 1970s, Codd @ IBM)
4. Object (???)

Notes: All need "tricks" to handle M-to-N rels.

A data model has (a) data structure and (b) operations

⇒ like an ADT.

Hierarchical Model: Background and Ideas

Background:

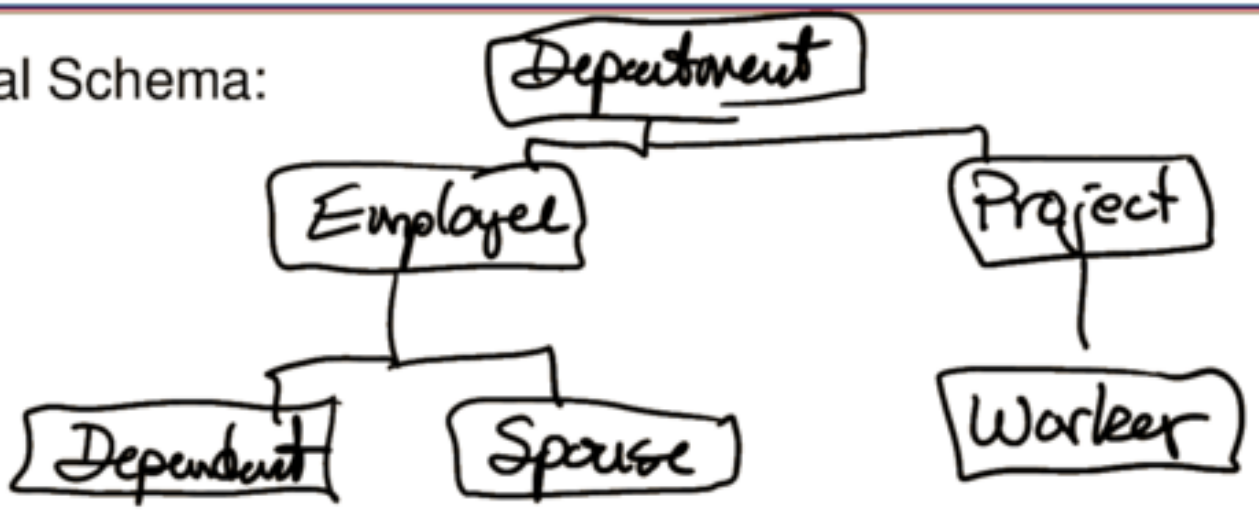
- John F. Kennedy, May 25, 1961: '. . . man on the moon . . . '
- Rockwell needed to organize parts for the Apollo CM & SM
- IBM created IMS (Information Management System) in 1968
  - original name: ICS/DL/I; thankfully renamed in '69
  - both used and introduced the Hierarchical Model
  - still sold today!

Ideas:

Schema is tree structured
only 1-N relationships are directly supported.
- 1 parent, many children.

Hierarchical Model: Terminology

Sample Logical Schema:

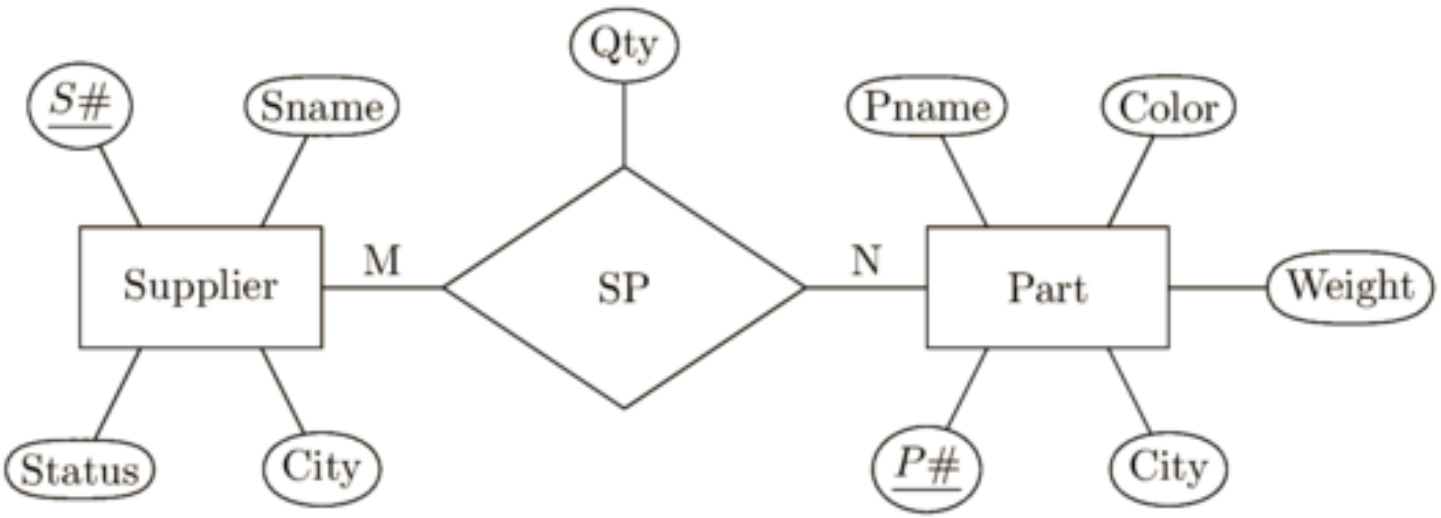


Terminology:

E-R Model		Hierarchical Model
Entity Set	≡	Segment Type
Entity	≡	Segment Occurrence (or Instance)
Attributes	≡	Fields or Data Items
Relationships	≡	PCRs - Parent-Child Relationships

Hierarchical Model: Supplier–Part Schema

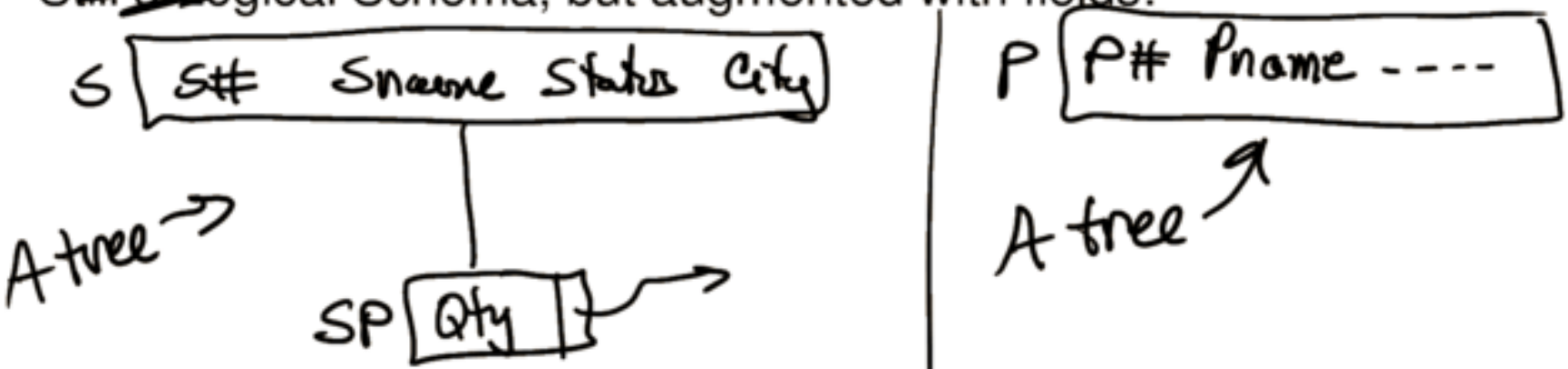
Consider this subset of Codd's SPJ schema:



– Don't need Project here.

Hierarchical Model: M:N Relationships (1 / 3)

~~Still a Logical~~ Physical Schema, but augmented with fields:

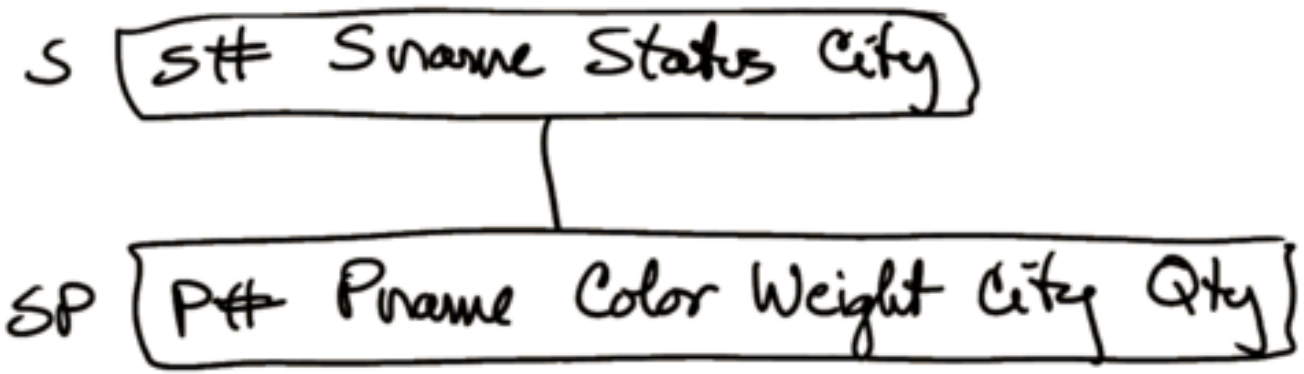


Notes: 2 separate trees.

The pointers from SP Segment Occurrences will point to seg. occurrences of P

Hierarchical Model: M:N Relationships (2 / 3)

Logical
~~Physical~~ Schema:

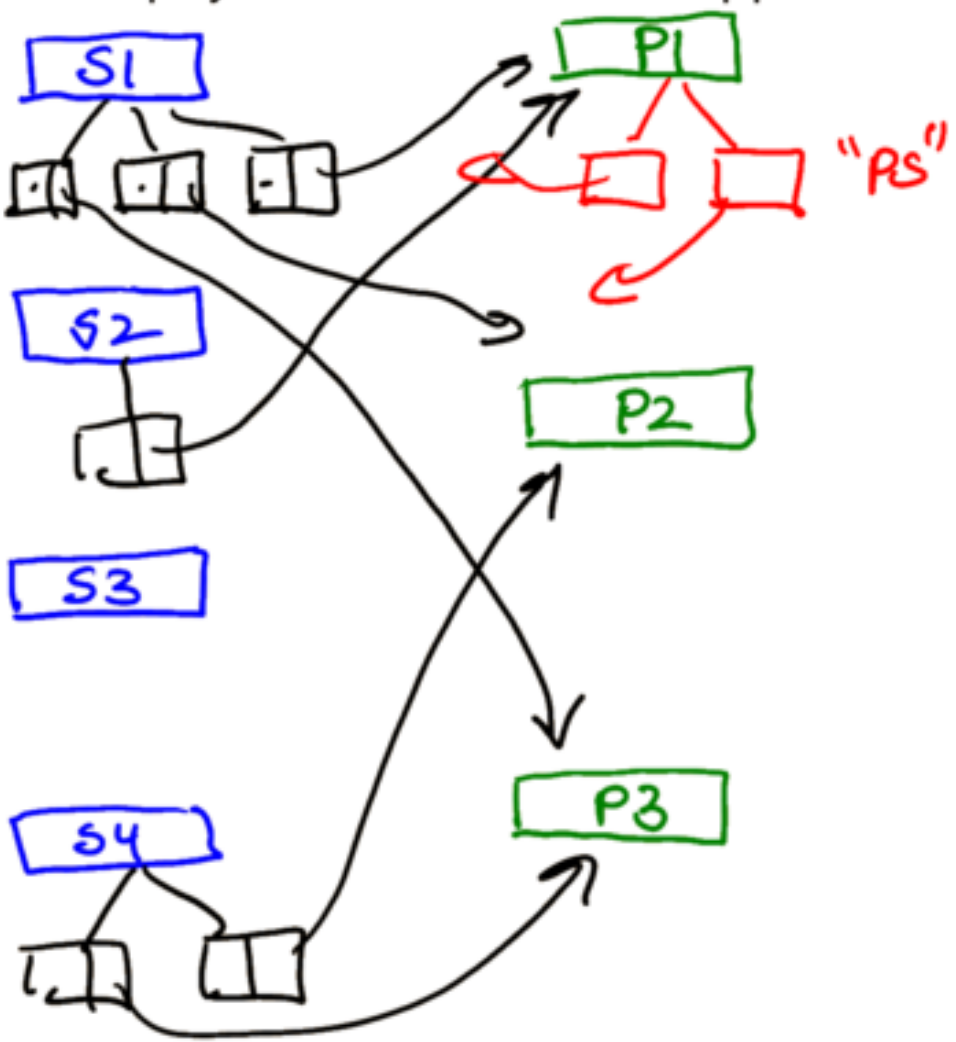


Notes: SP is the merger of the SP & P fields

It appears that if n suppliers supply a part, we will need n copies of that part.

Hierarchical Model: M:N Relationships (3 / 3)

The physical schema for Supplier–Part w/ sample data:



Questions:

1. Which parts can S1 supply?

2. Which suppliers can supply P3?

Network Model: Background and Ideas

- Created in the early 1970s by CODASYL's

  (Conference/Committee on Data Systems Languages)

  DBTG (Database Task Group)
- Goal: A standard theory of DB systems.

  Origin of the ideas of DML and DDL
- Became an ISO standard in 1987 (ISO 8907:1987)

  (And was withdrawn in 1998!)
- Graph–based instead of tree–based

Network Model: Terminology

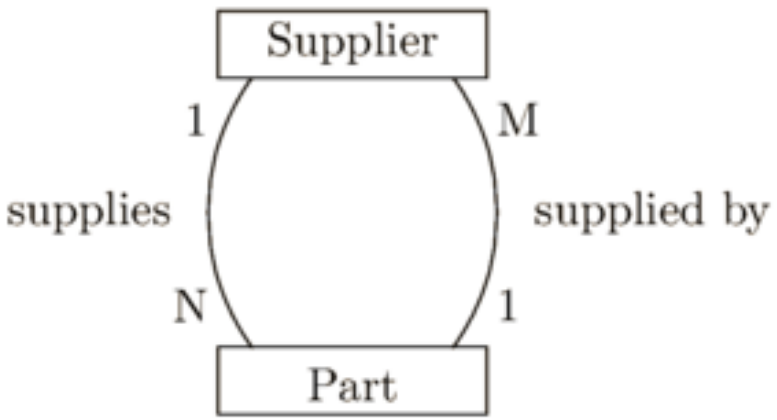
A Sample Logical Schema:

Terminology:

E-R Model		Network Model
Entity Set	≡	Record Type
Entity	≡	Record Occurrence
Attributes	≡	Data Items
Relationships	≡	Set or Link Type (orig. 'DBTG sets')

Network Model: M:N Relationships (1 / 2)

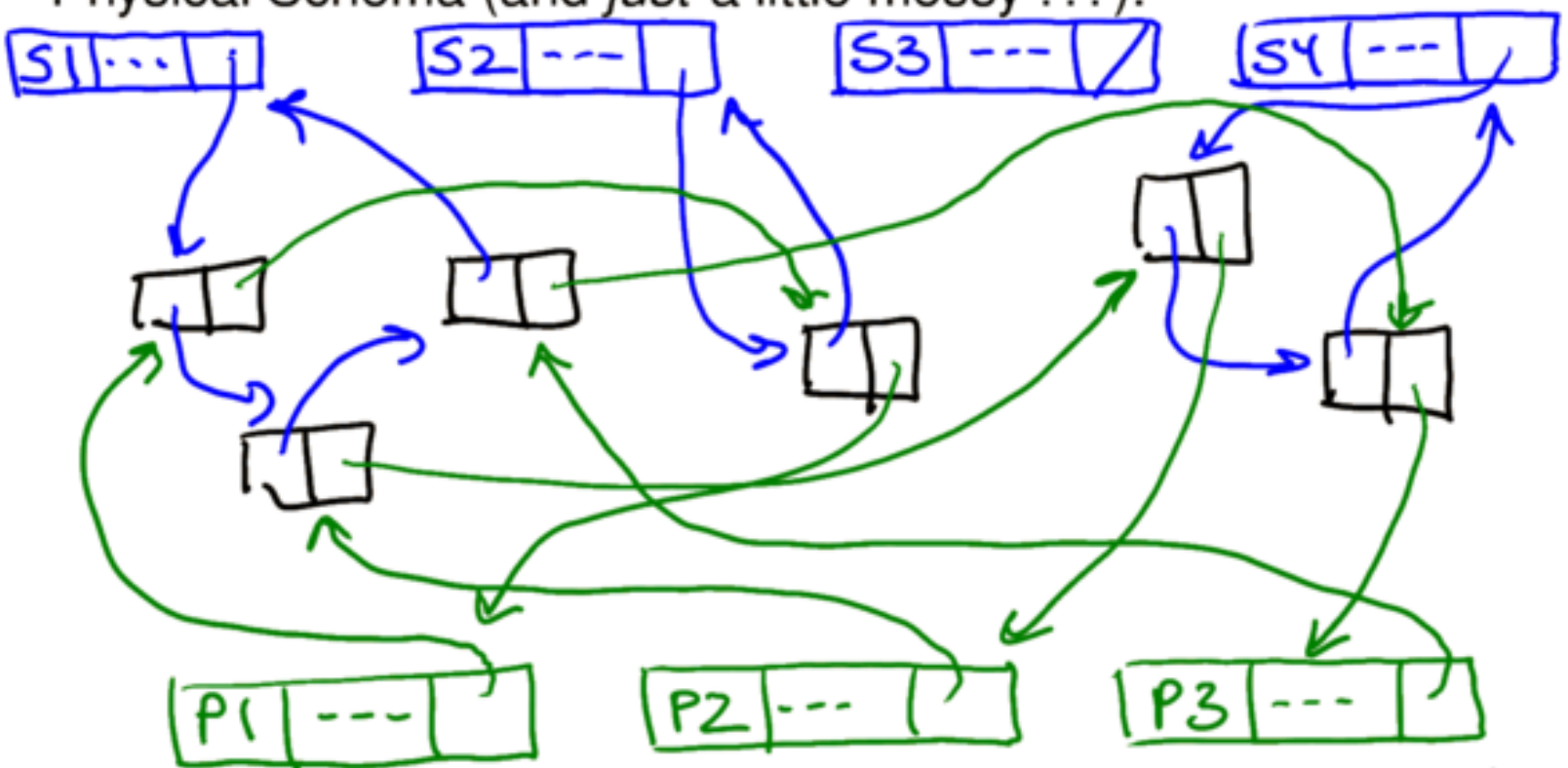
Logical Schema (of the M:N Supplier - Part relationship):



Note: An undirected graph – can travel either way.

Network Model: M:N Relationships (2 / 2)

Physical Schema (and *just* a little messy . . .):



Imagine the quantities are part of the Link Record Occurrences

① Which parts can S1 supply?

② Which suppliers can supply P3?

Relational Model: Background and Ideas

- Created by Edgar F. Codd. Famous paper:

  "A relational model of data for large shared data banks," 1970.

- Theoretical foundation: Set Theory
- Uses foreign keys instead of pointers
- No distinction between logical and physical schemas

Relational Model: DMLs

Codd proposed two types of DMLs:

1. Relational Calculus (non-procedural)
2. Relational Algebra (procedural)

SQL has elements of each

Watch for video!

Announcements

- Homework #1 is due now.
  - Late submissions can use at most one late day
  - Solutions will be posted sometime after 3:30pm tomorrow.
- Exam #1 is one week from today! (Oct 5th)
  - Topics: 1-5
- The video covering Topic 6 (Relational Calculus) is available in D2L (under Content/Lect. Recordings)
  - Today, we'll finish Topic 5 and will start Topic 7 (Relational Algebra).
- Homework #2 will be assigned Monday, due 9 days later.

Relational Model: DMLs

Partial Review

Codd proposed two types of DMLs:

1. Relational Calculus (non-procedural; "declarative")
   (Lecture Topic 6 – see video for coverage!)
2. Relational Algebra (procedural)
   (Lecture Topic 7)

Notes: Neither is suitable for practical use.

Codd's RC is now called Tuple RC, the later version is Domain RC.

Relational Model: Terminology (1 / 2)

Sample Supplier - Part Schema:

S

S#	Sname	Status	City
S1	Acme	10	Omro
S2	Fubar	10	Fisk
S3	Snafu	20	Ring

P

P#	Pname	Color	Weight	City
P1	Nut	Pink	0.2	Anton
P2	Bolt	Blue	0.9	Borea

Terminology:

E-R Model		Relational Model
Entity Set	≡	Relation (or table)
Entity	≡	Tuple (pronounce: TOO-ple)
Attributes	≡	Fields
Relationships	≡	Relationships!

Relational Model: Terminology (2 / 2)

Sample Department – Employee Schema:

PK

DEPARTMENT

DeptNum	DeptName	ManagerID	ManagerStartDate
1	Shipping	364	2001-04-01
2	Payroll	NULL	NULL
3	Billing	298	2000-11-17

FK

EMPLOYEE

Surname	GivenName	EmpNum	DeptID	Salary
Spade	Sam	786	1	48000
Trune	Joe	410	2	49500
Smith	Megan	364	1	75000
Maher	Mary	298	3	72000

Cardinality ≡ # of tuples

degree (or arity) ≡ # of attributes

unary relation ≡ A relation of Degree 1

n-ary relation ≡ A relation of Degree N

Relational Model: Misc. Notes

- Order of tuples in a relation is irrelevant (Why?) logically!

  Relations are sets of tuples!

- Fields are single-valued by default (vs. set-valued)

  This is frequently violated in practice.

- Relationships are supported by foreign keys

  May be stored in separate relations.
  (stay tuned!)

Relational Model: 1:N Relationships

We've already seen how to do this! (Just two slides ago!)

PK

DEPARTMENT

DeptNum	DeptName	ManagerID	ManagerStartDate
1	Shipping	364	2001-04-01
2	Payroll	NULL	NULL
3	Billing	298	2000-11-17

FK

EMPLOYEE

Surname	GivenName	EmpNum	DeptID	Salary
Spade	Sam	786	1	48000
Trune	Joe	410	2	49500
Smith	Megan	364	1	75000
Maher	Mary	298	3	72000

Notes: A dept can have many employees; each employee has 1 dept.

Relational Model: 1:1 Relationships

- Just a restriction of 1:N relationships:
  - We still store a FK in the 'many' relation
  - Must constrain the field's values to be unique; two options:

(1) Insist that the FK be a Candidate Key.

(2) Create a unique index on the FK
(the DBMS enforces the uniqueness
of the attr's content)

Relational Model: M:N Relationships

PK

S

S#	Sname	Status	City
S1	Acme	10	Omro
S2	Fubar	10	Fisk
S3	Snafu	20	Ring

PK

P

P#	Pname	Color	Weight	City
P1	Nut	Pink	0.2	Anton
P2	Bolt	Blue	0.9	Borea

FK FK

attr. of the relationship

SP

S#	P#	Qty
S1	P1	50
S1	P2	150
S2	P2	25
S3	P1	300

Notes:

- called a relationship relation
- necessary because won't work to store both FKs in source relations (try it!)

Object Model: Ideas

- OO programming languages have existed since Simula in 1967
- We'd like to be able to store objects in a DBMS
  - provides object persistence
  - can do it by mapping object instance fields to relational tuples, but that's clunky
- The Object Data Management Group (ODMG) defined an object-based DBMS standard
  - finished ODMG 3.0 in 2001 (and then disbanded!)

Object Model: Object DBMS Types

Two major varieties:

1. Object Oriented DBMS (OODBMS)
   - Marriage of an OOPL and a DBMS
2. Object Relational DBMS (ORDBMS)
   - A relational DBMS with added objects

Most popular
– get to leverage existing customer base.

Topic 6:

Relational Calculi

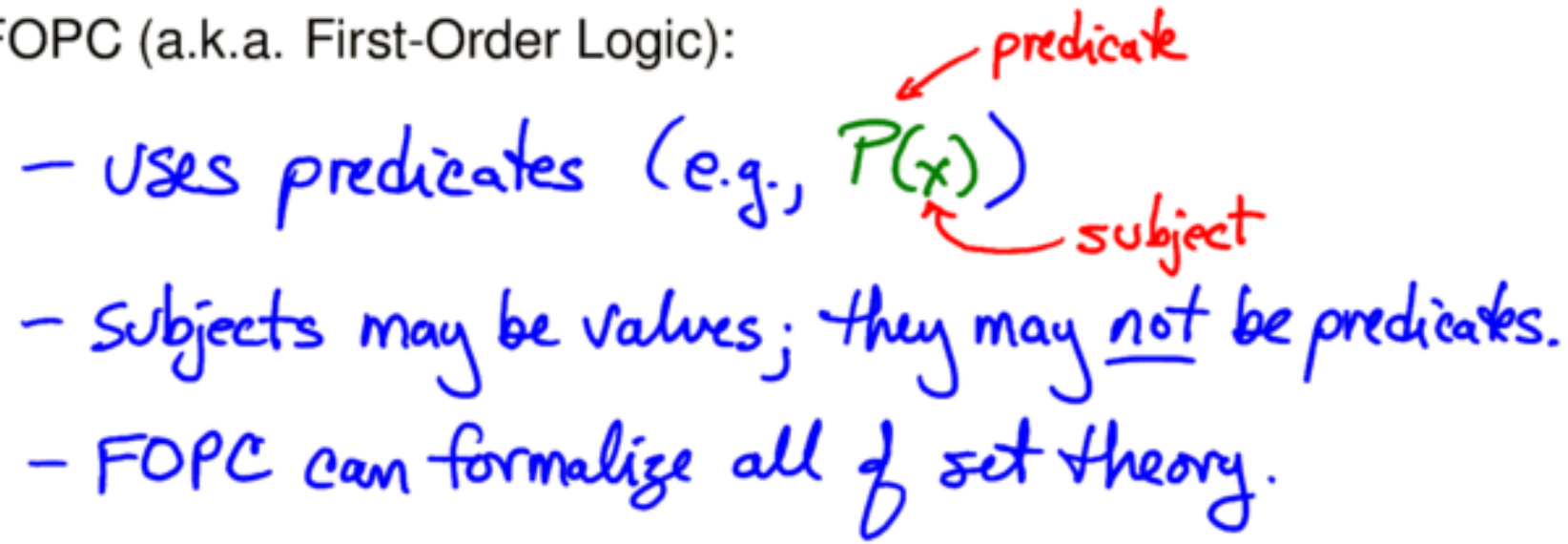
Meanings of 'Calculus'

- Calculus refers to any method or system of calculation
- 'Calculus' is derived from the Latin word for pebble
- Modern uses of the word include:
  - Differential Calculus: rates of change
  - Integral Calculus: limits of sums
  - Lambda Calculus: functional abstraction & application
  - Predicate Calculus: reasoning about symbolic logic

Review of First-Order Predicate Calculus (FOPC) (1 / 2)

Why? Because FOPC ⇒ Relational Calculus ⇒ SQL

FOPC (a.k.a. First-Order Logic):



Review of First-Order Predicate Calculus (FOPC) (2 / 2)

Supplied Primitives:

Variables, Logical operators, Quantifiers

Constructs of our creation:

Constants, Predicates, Functions

Example(s):

Feathers(x): x has feathers, $x \in$ Animal

Bird(x): x is a bird, $x \in$ Animal

Feathers(Eagle) $\rightarrow$ Bird(Eagle)

$\forall x$ [Feathers(x) $\Rightarrow$ Bird(x)], $x \in$ Animal

$\exists x$ [Bird(x) $\wedge \neg$Fly(x)], $x \in$ Animal

Relational Calculi: Ideas

Relational Calculi are "what, not how" languages

There are two forms:

Tuple Relational Calculus (TRC)

Domain " " (DRC)

Tuple Relational Calculus (TRC): Background

- Proposed by Codd in 1972
- So named because the variables in TRC represent tuples
- TRC queries have this basic form:

$$\{ t \mid \text{predicate}(t) \}$$

where:

t is a free (unbound) tuple variable

predicate is a wff

TRC: Query #1

Question: What is the content of the Employee Relation?

$$\{ e \mid Employee(e) \}$$

Recall these schemas:

DEPARTMENT	DeptNum	DeptName	ManagerID	ManagerStartDate	
EMPLOYEE	Surname	GivenName	EmpNum	DeptID	Salary

TRC: Query #2

Question: **What are the names and salaries of the people in department #5?**

$$\{ e.\text{givenname}, e.\text{surname}, e.\text{salary} \mid \text{Employee}(e) \wedge e.\text{deptid} = 5 \}$$

DEPARTMENT	DeptNum	DeptName	ManagerID	ManagerStartDate	
EMPLOYEE	Surname	GivenName	EmpNum	DeptID	Salary

= 5

TRC: Query #3

Question: **What are the names of the parts that can be supplied by individual suppliers in quantity > 200?**

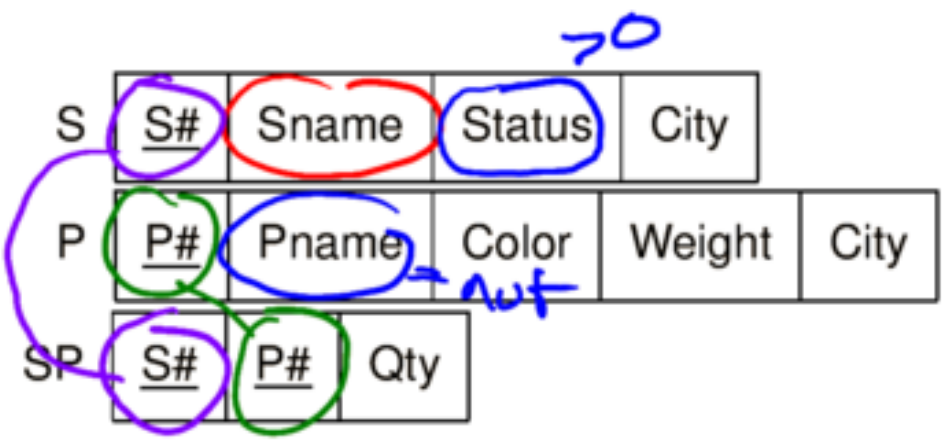
$$\{ p.pname \mid P(p) \wedge \exists(q)(SP(q) \wedge q.P\# = p.P\# \wedge q.Qty > 200) \}$$

S	S#	Sname	Status	City	
P	P#	Pname	Color	Weight	City
SP	S#	P#	Qty	$= 200$	

TRC: Query #4

Question: **What are the names of the active suppliers of nuts?**

$$\{ s.sname \mid S(s) \wedge s.status > 0 \wedge \exists(q)(SP(q) \wedge q.S\# = s.S\# \wedge \exists(p)(P(p) \wedge p.P\# = q.P\# \wedge p.pname = \text{'Nut'})) \}$$



Aside: Expression Safety

Definition: Expression Safety

An expression is safe if the expression's domain is the source of all resulting values.

Example(s):

Safe: $\{ e \mid Employee(e) \}$

Unsafe: $\{ e \mid \neg Employee(e) \}$

No way to prevent unsafe queries.

Domain Relational Calculus (DRC): Bkgd.

- Proposed by Lacroix and Pirotte in 1977 to supply a formalism for IBM's Query By Example (QBE) product.
- In DRC, a variable represents just one field of a tuple
- DRC queries have this basic form:

$$\{ \langle abcd \ldots \rangle \mid condition(\langle abcd \ldots \rangle) \}$$

where:

$\langle abcd \ldots \rangle$ is a set of ≥ 1 unbound domain variables, one per field of the relation

condition is a DRC w.f.f.

DRC Query #1

Question: What is the content of the Employee Relation?

$$\{ \langle efghi \rangle \mid \langle efghi \rangle \in Employee \}$$

Note the helpful field labels! ⇓

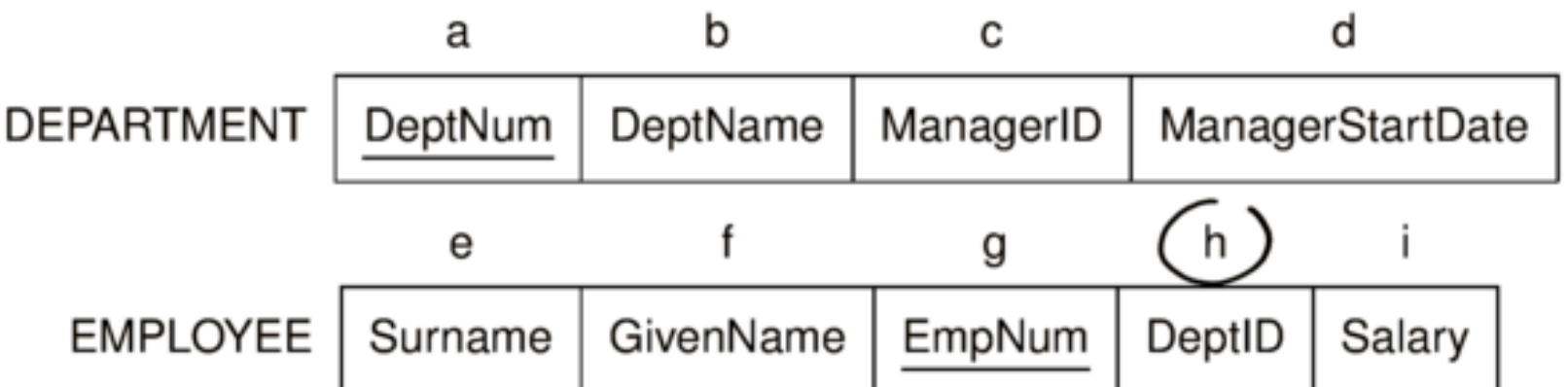
	a	b	c	d
DEPARTMENT	DeptNum	DeptName	ManagerID	ManagerStartDate

	e	f	g	h	i
EMPLOYEE	Surname	GivenName	EmpNum	DeptID	Salary

DRC Query #2

Question: **What are the names and salaries of the people in department #5?**

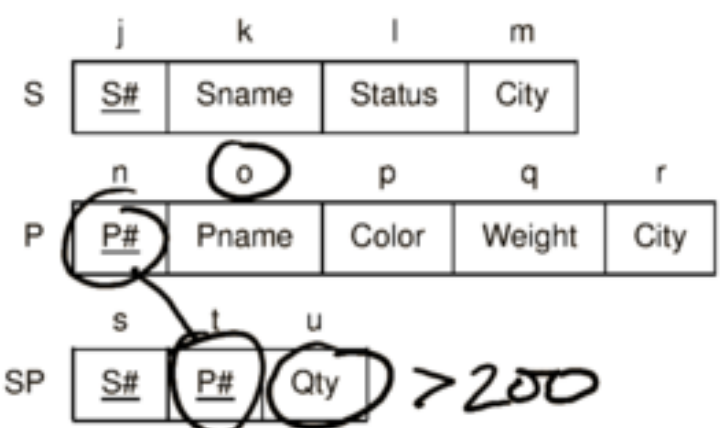
$$\{ \langle fei \rangle \mid (\exists h)(\langle efghi \rangle \in Employee \wedge h = 5) \}$$



DRC Query #3

Question: **What are the names of the parts that can be supplied by individual suppliers in quantity** > 200**?**

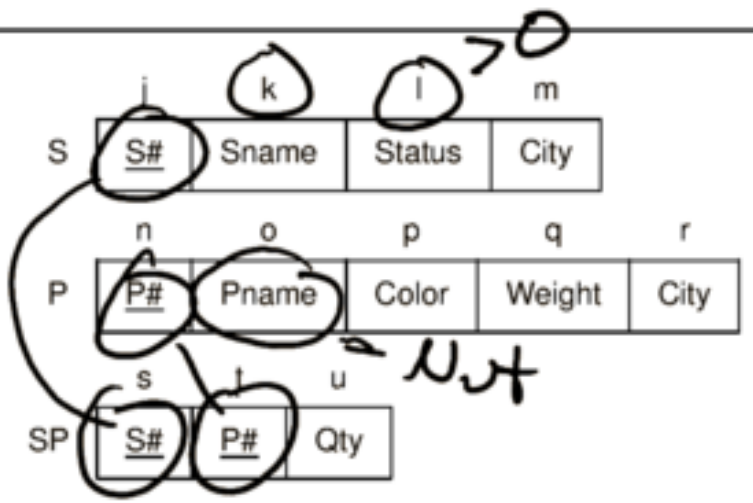
$$\{\langle o \rangle \mid (\exists n)(\langle nopqr \rangle \in P \wedge (\exists tu)(\langle stu \rangle \in SP \wedge n = t \wedge u > 200))\}$$



DRC Query #4 (1 / 2)

Question: **What are the names of the active suppliers of nuts?**

$$\{ \langle k \rangle \mid (\exists j\, l)(\langle j\, k\, l\, m \rangle \in S \wedge l > 0 \wedge (\exists s\, t)(\langle j\, t\, u \rangle \in SP \wedge j = s \wedge (\exists n\, o)(\langle t\, o\, p\, q\, r \rangle \in P \wedge n = t \wedge o = \text{'Nut'})))\}$$



Shortcut is optional.

Topic 7:

Relational Algebra

Looking for Topic 6 (Relational Calculus)? See:
- Video covering it (in D2L)
- Slides are in a separate file in the completed slides area of the class web page.

yes, Relational Calculus will be on Exam #2 and the Final Exam.

Background

- Introduced by Codd (along with the Tuple Relational Calculus)
- Relational Algebra . . . :
  - Is procedural, like most programming languages

    — we need to supply an ordering of operations
  - Would not be a good replacement for SQL in a DBMS
  - Is a good introduction to the operators provided by SQL

Relational Operators (1 / 2)

Relations are *closed* under Relational Algebra operators

- That is, they accept relations as operands, and produce relations as results.
- Example: Integers are closed under $+$ and $-$.

The eight basic Relational Algebra operators are:

$\times$	Cartesian Product	$\bowtie$	Join
$-$	Difference	π	Project
$\div$	Division	σ	Select
$\cap$	Insection	$\cup$	Union

Relational Operators (2 / 2)

The eight Relational Algebra operators can be grouped in two ways:

1. Set vs. Relational:
   - Set: $\cup$, $\cap$, $-$, $\times$, $\div$
   - Relational: σ, π, $\bowtie$
2. Fundamental vs. Derived:
   - Fundamental: σ, π, $\times$, $\cup$, $-$
   - Derived: $\cap$, $\bowtie$, $\div$

The Fundamental Operators

- Select (σ)
- Project (π)
- Cartesian Product ($\times$)
- Union ($\cup$)
- Difference ($-$)

Select (σ, sigma) (1 / 2)

- A unary (single argument) operator
- Chooses full tuples from a relation based on a condition
- Form: $\sigma_{condition}(relation)$

Example(s):

List all information of the employees in department #5:

$\sigma_{dept\# = 5}(Employee)$

Who are the active suppliers in Paris?

$\sigma_{city = 'Paris' \wedge status > 0}(S)$

Select (σ, sigma) (2 / 2)

Notes:

- Conditions may be as complex as is necessary
- Select is commutative:

$$\sigma_A(\sigma_B(\mathsf{r})) \equiv \sigma_B(\sigma_A(\mathsf{r}))$$

- Cascades of selects $\equiv$ conjunction in a single select:

$$\sigma_A(\sigma_B(\sigma_C(\mathsf{r}))) \equiv \sigma_{A \wedge B \wedge C}(\mathsf{r})$$

Project (π, pi) (1 / 2)

Pronunciation: PRO-ject (not pro-JECT, not PRAH-ject)

- Also a unary operator
- Chooses named columns from a relation
  - Resulting group of tuples may include duplicates . . .
  - . . . which we drop to preserve entity integrity
- Form:

Project (π, pi) (2 / 2)

Example(s):

Find the names & salaries of the employees in department 5:

$$\pi_{\text{givenname, surname, salary}}(\sigma_{\text{dept\#=5}}(\text{Employee}))$$

Alternatively:

$$\text{temp} \leftarrow \sigma_{\text{dept\#=5}}(\text{Employee})$$

$$\pi_{\text{givenname, surname, salary}}(\text{temp})$$

Cartesian Product ($\times$) (1 / 2)

$\{(x,y),(x,z)\ldots\}$

- A binary operator (form: R $\times$ S)
- 'Marries' all pairings of tuples from the given relations
  - resulting cardinality = card(R) $\cdot$ card(S)
  - resulting degree = degree(R) + degree(S)

Example(s):

A

m	n
2	i
3	iv
7	x

B

o	p
3	β
7	α

A x B

m	n	o	p
2	i	3	β
2	i	7	α
3	iv	3	β
3	iv	7	α
7	x	3	β
7	x	7	α

Cartesian Product ($\times$) (2 / 2)

Example(s):

What are the names of the active suppliers of nuts?

The complete query:

$$\pi_{\text{Sname}}(\sigma_{\text{Status}>0 \wedge \text{Pname='Nut'}}(\sigma_{\text{S.S\#} = \text{SP.S\#}}(\sigma_{\text{SP.P\#} = \text{P.P\#}}(S \times (SP \times P)))))$$

For a visualization of this query step–by–step with sample data, see the handout:

"Examples of the Relation Algebra Operations σ, π, and $\times$"

Announcements

- Exam #1 is Wednesday, in G-S 906.
  - Topics: 1-5 (DBs & DBMSes; Architectures; Files & Indexing; DB Design & E-R Model; Implementation Data Models)
  - Expect short-answer (1-3 sentences) and problem-solving questions, mostly.
  - You'll have the full 75 minutes.
- Handout: Homework #2 (Relational Algebra)

Union ($\cup$) (1 / 2)

- Another binary operator (form: R $\cup$ S)
- Result contains all tuples of both relations w/o duplicates

Example(s):

"Create a table of the Phoenix and Tucson employee data."

$PHXEmps \leftarrow \sigma_{loc='PHX'}(Employee)$

$TUSEmps \leftarrow \sigma_{loc='TUS'}(Employee)$

$AZEmps \leftarrow PHXEmps \cup TUSEmps$

Union (∪) (2 / 2)

To perform R ∪ S, R and S must be union compatible.

Definition: Union Compatible

Two relations that have the same degree and the same domains on corresponding pairs of attributes are union compatible

Example(s):

Can we legally union R(ssn, name) and S(age, hometown)?

yes, but ... why?

Difference (—) (1 / 2)

- Do you remember this one from basic sets?

$$\{a, b, c, d, e, f\} - \{b, d, f, h\} = \{a, c, e\}$$

- Yet another binary operator (form: R — S)
- Result is a relation of tuples from R that are not also in S
- Note that R and S must be union compatible

Difference (—) (2 / 2)

Example(s):

X

s	t
a	12
m	4
e	6

Y

u	v
e	6
a	16
f	4

X − Y

s	t
m	4
a	12

Y − X

u	v
a	16
f	4

not commutative!

The Derived Operators

- Intersection ($\cap$)
- Join ($\bowtie$)
- Division ($\div$)

Intersection (∩) (1 / 2)

- YABO — Yet Another Binary Operator! (form: R ∩ S)
- Resulting relation has the tuples that appear in both operand relations
- As with difference, R and S must be union compatible

Definition of ∩: $R \cap S \equiv R - (R - S)$

Intersection (∩) (2 / 2)

Example(s):

X

s	t
a	12
m	4
e	6

Y

u	v
e	6
a	16
f	4

X ∩ Y

s	t
e	6

$X - Y =$

s	t
a	12
m	4

$X - (X - Y)$

s	t
e	6

Join ($\bowtie$) (1 / 3)

- YABO! (form: R $\bowtie_{\text{condition}}$ S)
- Join is used to exploit PK–FK connections

  (using it with other attributes is unwise!)

Definition : $R \bowtie_\theta S \equiv \sigma_\theta (R \times S)$

Join ($\bowtie$) (2 / 3)

Example(s):

What are the names of the parts that can be supplied by individual suppliers in quantity > 200?

Without $\bowtie$: $\pi_{\text{pname}}(\sigma_{\text{qty}>200}(\sigma_{\text{SP.P\#}=\text{P.P\#}}(\text{SP} \times \text{P})))$

With $\bowtie$: $\pi_{\text{pname}}(\sigma_{\text{qty}>200}(\text{SP} \bowtie_{\text{SP.P\#}=\text{P.P\#}} \text{P}))$

Join ($\bowtie$) (3 / 3)

Three join variations:

1. Theta Join: $r \bowtie_\theta s$

   where θ is any PK-FK comparison ($=$, $\leq$, $\geq$, etc.)

2. Equijoin: $r \bowtie_\theta s$

   where θ is an equality PK-FK comparison

3. Natural Join: $r \bowtie s \equiv \pi_{R \cup S}(r \bowtie_{r.a1=s.a1 \wedge r.a2=s.a2 \wedge \ldots} s)$

   where R and S are the attribute sets of r and s, respectively

Division ($\div$) (1 / 9)

Let α and β be relations, where $A - B$ is the set difference of the attributes of α & β

Definition of Relational Division:

$$\alpha \div \beta = \pi_{A-B}(\alpha) - \pi_{A-B}((\pi_{A-B}(\alpha) \times \beta) - \alpha)$$

Notes:

- Famously hard to understand
- But you can't avoid it.

Division (÷) (2 / 9)

Let's review multiplication and division of integers:

Ex: $4 * 6 = 24$, so $24/6 = 4$ and $24/4 = 6$.

Now, consider Cartesian Product and Division with relations:

M

c
4
8

N

d
3
1
7

M × N = O

c	d
4	3
4	1
4	7
8	3
8	1
8	7

O ÷ N =

c
4
8

O ÷ M = N

Division (÷) (3 / 9)

What purpose does Division serve?

Division answers the "find X that are matched with all Y" queries.

Example(s):

Recall:

S

S#	Sname	Status	City

P

P#	Pname	Color	Weight	City

SP

S#	P#	Qty

X

Find the S#s of the suppliers that supply all parts of weight = 17.

Y

Division ($\div$) (4 / 9)

$$\alpha \div \beta = \pi_{A-B}(\alpha) - \pi_{A-B}((\pi_{A-B}(\alpha) \times \beta) - \alpha)$$

Example(s): (continued)

(Find the S#s of the suppliers that supply all parts of weight = 17)

"Find the X . . ." $\leftarrow$ X is S# (the matches we want)

". . . matched w/ all Y" $\leftarrow$ Y is the set of weight 17 P#s

Next, create the dividend (α) and divisor (β) relations:

S# P# P#

X Y Y

α | S# P# | β | P# |

$\alpha \leftarrow \pi_{S\#, P\#}(SP)$ | $\beta \leftarrow \pi_{P\#}(\sigma_{weight=17}(P))$

Division ($\div$) (5 / 9)

Example(s): (continued)

The content of α and β:

α

S#	P#
S1	P1
S2	P3
S2	P5
S3	P3
S3	P4
S4	P6
S5	P1
S5	P2
S5	P3
S5	P4
S5	P5
S5	P6

β

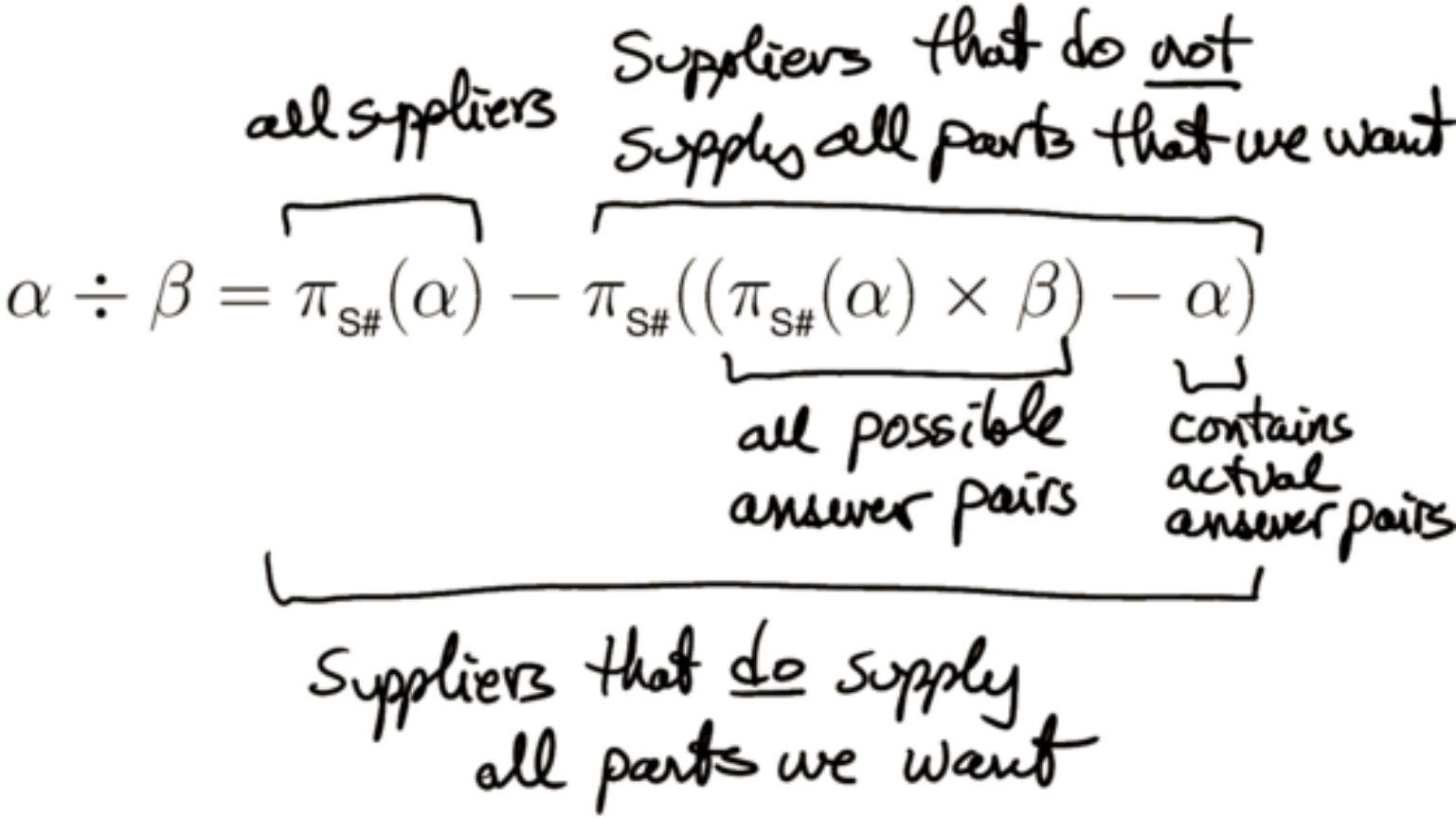
P#
P2
P3

Do any suppliers supply ALL of β's parts?

S5

Division (÷) (6 / 9)

Let's examine the definition in detail:



Division ($\div$) (7 / 9)

Looking at the data should help, too:

$\pi_{S\#}(\alpha) \times \beta$

S#	P#
S1	P2
S1	P3
S2	P2
S2	P3
S3	P2
S3	P3
S4	P2
S4	P3
S5	P2
S5	P3

α

S#	P#
S1	P1
S2	P3
S2	P5
S3	P3
S3	P4
S4	P6
S5	P1
S5	P2
S5	P3
S5	P4
S5	P5
S5	P6

$(\pi_{S\#}(\alpha) \times \beta) - \alpha$

S#	P#
S1	P2
S1	P3
S2	P2
S3	P2
S4	P2
S4	P3

Division ($\div$) (8 / 9)

And so, finally, we have our answer:

$\pi_{S\#}(\alpha)$

S#
S1
S2
S3
S4
S5

$\pi_{S\#}((\pi_{S\#}(\alpha) \times \beta) - \alpha)$

S#
S1
S2
S3
S4

$\alpha \div \beta = \pi_{S\#}(\alpha) - \pi_{S\#}((\pi_{S\#}(\alpha) \times \beta) - \alpha) =$

S#
S5

Supplier S5 supplies all parts of weight 17 (P2 and P3).

Division ($\div$) (9 / 9)

Is there a short-cut to avoid that mess? **NO!**

:-O

Consider this very similar query:

> Find the S#s of the suppliers that supply
>
> all parts of weight = **19**.

The only weight = 19 part is P6, which this simple expression produces: $\pi_{\mathsf{S\#}}(\alpha \bowtie_{\mathsf{P\#}} \beta)$ ($\rightarrow$ S4 and S5)

But, when weight = 17, that query gives S2, S3, and S5!

Announcements

- Reminders:
  - Exam #1's new date is TBD
  - Homework #2's due date is TBD
- The next assignment will be Homework #3, on SQL queries.
- Today: • Reset the dates for EX 1 & HW 2
  • Start Topic 8: SQL
- And watch the Topic 6 video, if you haven't already.

Topic 8:

Structured Query Language (SQL)

Background (1 / 2)

- IBM's System R was released in 1978
  - Its query language name: SEQUEL
    (Structured English QUEry Language)
  - But trademarked by a British airplane company!
    (1982, Hawber Siddeley Dynamics Engineering Ltd.)
  - After dropping the vowels: SQL
- IBM's current DB/2 was released in 1982; also used SQL
- SQL:
  - A marriage of TRC to RA
  - SQL = DML + DDL + DCL + QL + . . .

Background (2 / 2)

- SQL is no longer a proprietary language:
  - SQL is now an ANSI/ISO standard (ISO/IEC 9075)
  - Versions: 1989, '92, '99, '03, '06, '08, '11, '16, ...
- But no DBMS strictly follows any of them!
  - Example: Tuple IDs are non–standard
  - There is a basic subset you can count on

Relational Operators (1 / 5)

But first: SQL's SELECT statement

- NOT identical to the select operator of Rel. Alg.!
- Most basic Form:

```
SELECT <attribute list>
FROM   <relation list>;
```

Example(s): Display the names & statuses of the suppliers

```
select sname, status
  from s;
```

Relational Operators (2 / 5)

Now that we can perform π, we can answer our first standard query:

> "What is the content of the Employee relation?"

```
select surname, givenname, empnum, deptid, salary
from employee;

    or

select *
from employee;
```

Relational Operators (3 / 5)

Performing σ requires a new clause:

```
SELECT <attribute list>
FROM   <relation list>
WHERE <condition> ;
```

Example(s):

What are the names and salaries of employees in department 5?

```
select givenname, surname, salary
 from employee
 where deptid = 5 ;
```

$\pi_{gn,sn,s}(\sigma_{deptid=5}(employee))$

Relational Operators (4 / 5)

These are also all of the clauses that we need for ⋈:

Example(s):

What are the names of the parts that can be supplied by individual suppliers in quantity > 200?

pname

qty

```
select  pname
from    p, spj
where   p.pno = spj.pno
  and   qty > 200;
```

Announcements

- Exam #1 is now a take-home
  - To be released tomorrow @ noon (crossed fingers...)
  - Due Sunday @ 11:59pm
  - Topics now 1 through 6.
- Homework #2 is now due Wed Oct 19th @ the start of class.
- Homework #3 (SQL queries) will be assigned after we've covered enough SQL for you to do it.

Relational Operators (4 / 5)

These are also all of the clauses that we need for ⋈:

Example(s):

What are the names of the parts that can be supplied by individual suppliers in quantity > 200?

Using an "old syntax" join:

```
select pname
from P, SPJ
where P.pno = SPJ.pno and qty > 200;
```

Using a more recent syntax:

```
select pname
from P join SPJ on (P.pno = SPJ.pno)
where qty > 200;
```

Relational Operators (5 / 5)

For completeness, our fourth standard query:

Example(s):

What are the names of the active suppliers of nuts?

```
SELECT sname
  FROM s, spj, p
 WHERE s.sno = spj.sno
   AND spj.pno = p.pno
   AND status > 0 AND pname = 'nut';
```

sname

pname

join conditions

select condition

Alternative #1:

```
select sname
From s join spj on (s.sno = spj.sno)
     join p on (spj.pno = p.pno)
where status > 0 and pname = 'nut';
```

parens opt.

Alternative #2:

```
select ---
From s join spj using (sno)
     join p using (pno)
where --- ;
```

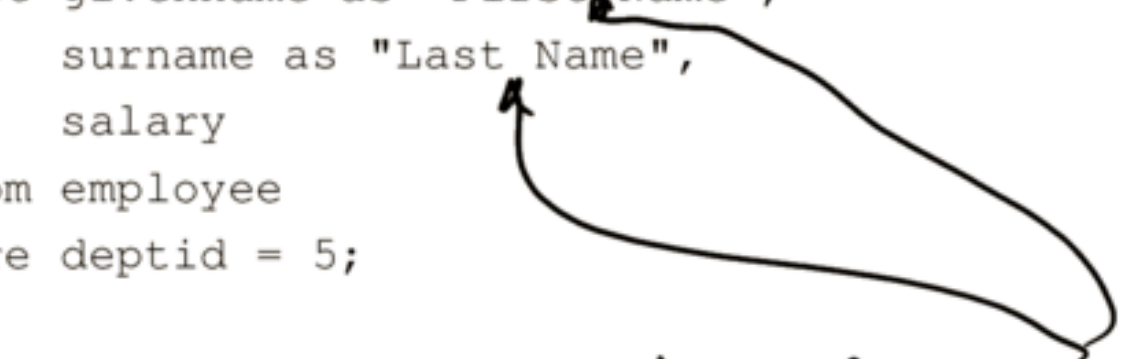
parens required!

Column Aliases

You may give your result relations different attribute names:

Example(s):

```
select givenname as "First Name",
       surname as "Last Name",
       salary
  from employee
 where deptid = 5;
```



double-quotes needs b/c of the spaces

A Note about Duplicate Tuples

By default, SQL does not remove duplicate tuples from result relations. (Why not? It should, relations are sets!)

(1) Efficiency – removing duplicates costs!

(2) Duplicates can have meaning for the user.

But SQL lets us override that behavior!

Example(s):

```
select distinct sno
from spj j
```

Ordering Result Tuples

We can sort tuples, too, with the ORDER BY clause.

Example(s):

Ascending Order:

```
select sname, city
  from s
order by city;
```

Descending Order:

```
select sname, city
  from s
order by city desc;
```

We can even do "phone book" sorting:

```
select *
from spj
order by pno, sno;
```

Computed Columns

We can perform basic arithmetic with field values:

Example(s): Convert part weight from pounds to grams:

```
select pname, weight * 454 as "weight(grams)"
  from p;
```

Tuple Aliases

We can assign relations temporary, alternate names.

Example(s):

in SQL std, "as" is optional
in Oracle, "as" is forbidden here.

Create all pairs of supplier names located within the same city:

```
select first.sname, second.sname
  from s as first, s as second
  where first.city = second.city ;
```

London, Paris and Paris, London

same as above, w/ one new line:

```
  -- and first.sno < second.sno;
```

Pattern Matching (1 / 2)

SQL allows us to search for values that match a particular pattern.

Form:

. . . **WHERE** *attribute* [not] **LIKE** '*pattern*'
[ESCAPE *escape character*]

Available wildcards:

_ (underscore) matches any single character

% matches 0 or more characters

Important: LIKE does not support regular expressions.

Pattern Matching (2 / 2)

Example(s):

Find the part names that have an ‘o’ as the second letter:

```
select pname
from   p
where  pname like '_o%';
```

To use wildcards as regular characters, ESCAPE them:

```
... where field like '%@%' escape '@';
```

Here, we match any string ending in a percent sign.

Regular Expressions (1 / 2)

Oracle offers REGEXP_LIKE for regular expressions.

Form (note that <*pattern*> and <*match*> are single–quoted):

... **WHERE REGEXP_LIKE** (<*source*>, '<*pattern*>', '<*match*>');

where:

<*source*> is an attribute name
<*pattern*> is a regular expression (see next slide)
<*match*> is a search modifier; e.g.:

- `c` — case sensitive
- `i` — case **in**sensitive
- `x` — ignore whitespace
- ⋮

Regular Expressions (2 / 2)

REGEXP_LIKE options for <*pattern*> include:

.	—	(a period) match a single character
x*	—	match **x** 0 or more times
x+	—	match **x** 1 or more times
x?	—	match **x** 0 or 1 times
x|y	—	match **x** once or match **y** once
x{n,m}	—	match **x** at least **n** times, at most **m** times

Example(s):

Find the part names that have an 'o' as the second letter:

```
select pname
from   p
where  regexp_like( pname , '.o()*' , 'i' );
```

or '.o.*'

Set Operators (1 / 5)

Cartesian Product ($\times$):

- Cartesian Product produces all pairs of tuples.
- Join produces all pairs of tuples that meet a condition.
- So . . . if we Join when the condition is always true . . .

Example(s): To form the Cartesian Product of S and P:

```
select *
from s, p j
```

Set Operators (2 / 5)

Union ($\cup$):

- Form: select ...
  union [all] (all $\equiv$ keep duplicates)
  select ...
- Union compatibility still applies!

Example(s):

```
select city from s
 union all
select city from p;
```

Default: $\cup$ follows set theory, σ does not!

Set Operators (3 / 5)

Intersection ($\cap$) and Difference ($-$):

- The SQL keyword for set intersection is INTERSECT
- The SQL keyword for set difference is EXCEPT

  ... except, Oracle uses MINUS
- Form: select ...
  intersect/except
  select ...

Example(s):

```
select city from s
EXCEPT                          <-- MINUS in Oracle
select city from p;
```

Set Operators (4 / 5)

The Return of . . . Division!

Version 1: Relational Algebra expression

Recall: $\alpha \div \beta = \pi_{\text{A-B}}(\alpha) - \pi_{\text{A-B}}((\pi_{\text{A-B}}(\alpha) \times \beta) - \alpha)$

And our sample division query:

> "Find the S#s of the suppliers who supply all parts of weight equal to 17."

Set Operators (5 / 5)

And so, $\alpha \div \beta = \pi_{\text{A-B}}(\alpha) - \pi_{\text{A-B}}((\pi_{\text{A-B}}(\alpha) \times \beta) - \alpha)$

becomes in SQL:

```
select distinct sno from spj
except
select sno from
       ( select sno, pno
           from (select sno from spj) as t1,
                (select pno from p where weight=17) as t2
         except
         select sno, pno from spj
       ) as t3;
```

Notes:

- duplicate tuples in operands won't confuse EXCEPT
- sub-SELECTs in FROMs need table aliases (e.g. t1)
- drop "as" in Oracle
  - and change except to minus in Oracle, too!

Aggregate Functions (1 / 3): Background

Idea: Let SQL compute basic statistical results for us

SQL provides aggregate functions for this purpose:

- **count([distinct] *attr*)** — counting entries in a relation
- **sum([distinct] *attr*)** — totaling values of *attr* in a relation
- **avg([distinct] *attr*)** — averaging values of *attr* in a relation
- **min(*attr*)** — smallest value of *attr* in a relation
- **max(*attr*)** — largest value of *attr* in a relation

Aggregate Functions (2 / 3)

Example(s): Variations on counting:

- `select count(city) from p;`

  ⇒ 6 (duplicates are counted) [count values]

- `select count(distinct city) from p;`

  ⇒ 3 (removes duplicates)

- `select count(*) from p;`

  ⇒ 6 (counts tuples) – reports cardinality

⇒ count(*) counts null tuples; count(field) does not.

Aggregate Functions (3 / 3)

Example(s):

If we have one of each part in a box, how much does the content weigh?

Which query will give the correct answer?

(a) `select sum(weight) from p;`

(b) `select sum(distinct weight) from p;`

Correct! Don't want to eliminate copies.

Announcements

- Exam #1:
  - Everyone submitted something!
  - We will grade it as soon as we can, will be done before the 10/30 withdrawal deadline!
- Homework #2 is due
- Handout: Homework #3, due next Monday (Oct. 24)
- Program #3 proposal:
  - I'll reduce the work, you accept 5 days of overlap with Homework 3. So, assigned Wed 10/19, due Wed 11/2. Deal?

Group By

Purpose: Apply aggregates to sub–groups of tuples

Example(s):

What are the average quantities in which suppliers are supplying parts?

```
select sno, avg(qty)
 from spj
group by sno;
```

Having

- Used in conjunction with 'group by'
- Purpose: Controls which group's aggregations are produced

Example(s):

Which suppliers are supplying parts in average quantity under 400, and what are those averages?

```
select sno, avg(qty)
  from spj
group by sno
  having avg(qty) < 400;
```

More on Nested Queries (1 / 4)

We've seen this idea before (e.g., the division query).

Another way to do nested queries is with the IN operator:

- IN tests set membership (form: tuple IN relation)
- We can negate the test (tuple NOT IN relation)
- Used in conjunction with a sub–query in a WHERE clause

Example(s):

Remember this query?

```
select pname
 from p, spj
 where p.pno=spj.pno and qty>200;
```

More on Nested Queries (2 / 4)

Example(s):

Idea: Create a set of parts available in quantity > 200,

and test each part from the DB against that set.

To create the P#s of the 'quantity > 200' parts:

```
select pno
from   spj
where  qty > 200;
```

And to produce the names of the parts in that set:

```
Select pname
from P
where pno in (select pno
              from spj
              where qty > 200);
```

More on Nested Queries (3 / 4)

Notes:

- IN and NOT IN are only suitable for equality comparisons
- Other options include:
  - > any (meaning: "greater than some")
  - < any (" less " "
  - = any (meaning: IN)
  - > all (obvious!)

More on Nested Queries (4 / 4)

One more nested–query operator: EXISTS

Its purpose: Test if a relation holds at least one tuple

Example(s):

Another (awkward!) version of the qty > 200 query:

```
select pname
  from p
  where exists ( select *
                   from spj
                   where p.pno = spj.pno
                     and qty > 200 );
```

Division, Revisited (1 / 7)

Version 2: "Double $\neg\,\exists$"

Recall:

> Find the S#s of the suppliers who supply all parts of weight 17.

Restated in logical English:

> Find S#s such that $\forall$ parts of weight 17, $\exists$ suppliers that supply them all

Apply Double Negation and Generalized De Morgan's Laws:

$\neg\,\neg\ \forall a\,\exists b\,f(a,b) \equiv \neg\,\exists a\,\neg\,\exists b\,f(a,b)$

Returning to logical English:

> Find S#s such that $\neg\,\exists$ parts of weight 17 for which $\neg\,\exists$ suppliers that supply them all

Division, Revisited (2 / 7)

Find S#s such that $\neg \exists$ parts of weight 17 for which $\neg \exists$ suppliers that supply them all expressed in SQL:

```
select distinct sno
from   spj as global
where  not exists
    ( select pno
      from   p
      where  weight = 17 and not exists
          ( select *
            from   spj as local
            where  local.pno = p.pno
              and  local.sno = global.sno
          )
    )
```

Division, Revisited (3 / 7)

Aside: This query form is useful beyond division.

Example(s): Which drinkers like a unique set of beers?

Nesting Depth

```
0  SELECT  L1.drinker
   FROM    Likes L1
   WHERE   NOT EXISTS(
1            SELECT  *
             FROM    Likes L2
             WHERE   L1.drinker <> L2.drinker
             AND     NOT EXISTS(
2                      SELECT  *
                       FROM    Likes L3
                       WHERE   L3.drinker = L2.drinker
                       AND     NOT EXISTS(
3                                SELECT  *
                                 FROM    Likes L4
                                 WHERE   L4.drinker = L1.drinker
                                 AND     L4.beer = L3.beer))
             AND     NOT EXISTS(
2                      SELECT  *
                       FROM    Likes L5
                       WHERE   L5. drinker = L1. drinker
                       AND     NOT EXISTS(
3                                SELECT  *
                                 FROM    Likes L6
                                 WHERE   L6.drinker = L2.drinker
                                 AND     L6.beer= L5.beer)))
```

Schema:

Likes(drinker , beer)

Source: Leventidis, A., et. al. "QueryVis: Logic–based Diagrams help Users Understand Complicated SQL Queries Faster." Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, June 2020, pp. 2303–2318. https://doi.org/10.1145/3318464.3389767

Division, Revisited (4 / 7)

Version 3: Set Containment

Observation:

If $B \subseteq A$, then $B - A$ will be empty (or, $\neg\exists(B - A)$)

Relevance:

If a supplier supplies a superset of the parts

of weight 17, the supplier clearly supplies them all

A = The parts a supplier supplies

B = The parts of weight 17

Division, Revisited (5 / 7)

```
select distinct sno
from   spj as global
where  not exists (                  --   not bkwd-E
    ( select pno
      from   p                       --   B
      where  weight = 17
    ) except (                       --   minus
      select p.pno
      from   p, spj                  --   A
      where  p.pno = spj.pno
        and  spj.sno = global.sno
    )
)
```

Division, Revisited (6 / 7)

Version 4: Set Cardinality

Idea:

- For each supplier that supplies parts of weight 17, count those parts.
- If the total matches the number of weight 17 parts, that supplier supplies them all.

Division, Revisited (7 / 7)

```
select   distinct sno
from     spj, p
where    spj.pno = p.pno and weight = 17
group by sno
having   count(distinct p.pno) =
            ( select count (distinct pno)
              from   p
              where  weight = 17
            )
```

Outer Joins (1 / 5)

Regular ("inner") joins discard non-matching tuples.

Example(s): Name the employees who are supervising buildings.

M

Id	Name
1	Roy
2	Amy
3	Joy

N

Building	Supervisor
A	2
B	1
C	2
D	NULL

$M \bowtie_{id=supervisor} N$

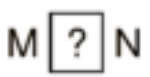
Id	Name	Building	Supervisor
2	Amy	A	2
1	Roy	B	1
2	Amy	C	2

Outer Joins (2 / 5)

Now consider this slightly different query.

Example(s): Name all employees and the buildings they supervise.

Our desired answer:



Id	Name	Building	Supervisor
1	Roy	B	1
2	Amy	A	2
2	Amy	C	2
3	Joy	NULL	NULL

But . . . how do we get this result from a join?

Outer Joins (3 / 5)

There are three varieties of outer join:

- Left Outer Join (): Retains unmatched tuples from left relation
- Right Outer Join () Retains unmatched tuples from right relation
- Full Outer Join (): Retains all unmatched tuples

Outer Joins (4 / 5)

The SQL outer join syntax:

select <*attribute list*>

from (<*relation*> **[left/right/full] outer join** <*relation*> **on** <*join condition*>)

where <*condition*> ;

Example(s): Name all employees and the buildings they supervise.

```
select *
from ( m left outer join n on m.id=n.supervisor );
```

Outer Joins (5 / 5)

Outer join is not an fundamental operator.

We can fabricate outer join with UNION ALL.

Example(s): Name all employees and the buildings they supervise.

```
select id, name, building, supervisor
from   m, n
where  m.id = n.supervisor
```

union all

```
select id, name, NULL, NULL
from m
where id not in (select supervisor
                 from n
                 where supervisor is not null);
```

SQL as DDL

First order of business: Creating a database!

The exact mechanism depends on the DBMS.

1. Postgres: `$ createdb <name>`
2. Oracle: `CREATE DATABASE <name>;`

Creating Relations (1 / 3)

Some sample attribute types:

- Integers: `integer`, `number(p)`
- Floats: `float`, `real`, `number(p,s)`
  - `p` is precision (total # digits), `s` is scale (# digits after decimal)
- Strings: `char(n)`, `varchar(n)`, `varchar2(n)`
- Others: `timestamp`, `blob`, `bfile`, ...

- In varchar2(n), null = empty string
  Someday, varchar(n) will be different
  Today, varchar(n) ≡ varchar2(n).
  ⇒ Use varchar2!

Creating Relations (2 / 3)

To create a relation:

CREATE TABLE <*table name*> (

<*attribute name*> <*data type* > [NOT NULL],

. . .

[PRIMARY KEY (<*attribute*>)]

);

- use normal variable naming rules
- if you id an attr as a primary key, you'll get a primary index on it.
- your DBMS will have a LOT of other options!

Creating Relations (3 / 3)

Example(s):

Creating the supplier (S) relation:

```
create table s (
    sno     varchar2(5),  -- the supplier ID number
    sname   varchar2(20), -- the supplier's name
    status  integer,      -- supplier status
    city    varchar2(15), -- location of supplier
    primary key (sno)
);
```

Additional Scribblings (1)

Week	Monday	Wednesday	
9	Assign H3 17	Assign P3 19	
10	H3 due 24	26	
11	31	P3 due 2 Assign H4	
12	7	H4 due 9 Assign P4	
13	14	Exam #2 16	
14	P4 duedate 1 21	23	Thanksgiving

Announcements

- Homework #2 is due now!
  - Can use a late day if you need to do so.
- Homework #3 (SQL Queries) is due Monday.
  - Already have two submissions!
- Handout: Program #3 (JDBC)
  - As we agreed Monday, this has fewer/easier queries in exchange for overlapping by 5 days with Homework #3.
- Today: Jump ahead to Topic 9 (JDBC), and if there's time jump back to Topic 8 (SQL).

Inserting Tuples into a Relation

To insert a tuple into a relation:

INSERT INTO <*relation name*> [(<*column list*>)]

VALUES (<*expression list*>);

Example(s):

```
Insert into foo values ('z', 1, 'b');
```

- assumes attrs are given in the same order that they are listed in the CREATE TABLE command.
- `set autocommit off;` will make a seq. of insertions go faster.

Topic 9:

SQL in Applications

Classic Approaches

1. Use a preprocessor
   - Usually for older languages (e.g., C and C++)
   - I'll show an example or two of this, just for context
2. Use a library (API)
   - Usually the only option for languages with APIs
   - Often several options per language

The Preprocessor Approach (1 / 2)

A common C program line: `#include <stdio.h>`

But that is not C; rather, it's a preprocessor directive

A preprocessor can be used to expand DBMS commands, thus saving us coding:

1. Insert preprocessor statements into program code
2. Execute the DBMS' preprocessor
3. Compile & link the program
4. Execute the application

The Preprocessor Approach (2 / 2)

Two varieties of preprocessed SQL statements:

1. Embedded SQL
   - SQL statements are hard–coded (static)
2. Dynamic SQL
   - Arguments added to an SQL statement shell

Cursors

A Problem:

How large will your query's result be?

(That is, how much memory do we need to hold what the DBMS is going to return to us?)

The Solution: Just fetch one tuple at a time! (Let the DBMS and your OS handle the buffering.)

– Cursors can be implicit or explicit

implicit ↑ API (usually)

explicit ↑ preprocessor

Preprocessor Examples

See the Sample C & Postgres programs on class webpage!

"Embedded and Dynamic SQL APIs in Postgres"

Advantage:

- Can be adapted to any programming language

Disadvantages:

- Several preprocessor directives to learn.
- Very little abstraction (e.g., cursors are explicit)

The Library Approach

Advantage:

- Just another API; use it like any other API

Disadvantage:

- Might be a 3rd party add–on; needs to be installed

ODBC vs. JDBC

ODBC:

- An early 1990s Microsoft API to connect C programs to DBMSes
- ODBC stands for "Open Database Connectivity"
- Recently (2018) updated by Microsoft to support hierarchical and semistructured data

JDBC:

- Sun Microsystem's (now Oracle's) 1997 Java API based on ODBC
- JDBC stands for ... JDBC !

JDBC

Core capabilities:

1. Est. connection to a data source
2. Send SQL stmts to the source
3. Process returned results and messages

Some related technologies:

- SQLJ — a preprocessor–based Java language extension
- Java Persistence API (JPA) — supplies object persistence
- Java Data Objects (JDO) — ditto

Using JDBC (1 / 4)

1. Establish connection to a data source

(a) `import java.sql.*`

(b) Load the driver (names vary by DBMS)

- Add your DBMS' JAR file to your classpath:
  - Oracle 11 via lectura: **`ojdbc8.jar`**
- Call `Class.forName()` to initialize the driver class:
  - Oracle 11: **`oracle.jdbc.OracleDriver`**

Using JDBC (2 / 4)

(c) Connect to the DBMS

```
Connection dbConnect = DriverManager.getConnection (
    "jdbc:oracle:thin:@host.foo.bar.com:1234:oracle",
    "username", "password" );
```

where:

- The first argument is the DB URL. Parts:
  - `thin` is the type of driver
  - `host.foo.bar.com` is the DBMS server
  - `1234` is the port number
  - `oracle` is the sid (system ID)
- `username` is the user's DBMS login
- `password` is the user's DBMS password

Using JDBC (3 / 4)

2. Send SQL statements to that source

Create a Statement object:

```
Statement stmt = dbConnect.createStatement();
```

Ask it to execute the SQL query:

```
ResultSet answer = stmt.executeQuery (
    "SELECT sno, status FROM s" );
```

NOTE: No semicolon after the query!

Using JDBC (4 / 4)

3. Process returned results and messages

JDBC uses cursors, too, but the details are implicit.

Before the first read, test `answer.next()`:

> If true, a tuple is available

Then, fetch field values by type. E.g.:

```
answer.getString("sno")
answer.getInt("status")
```

Accessing MetaData with JDBC

First, get a `ResultSetMetaData` object by calling:

```
rsmd = answer.getMetaData()
```

Then, fetch the metadata you want to see. E.g.:

`rsmd.getColumnCount()`	returns degree
`rsmd.getColumnName()`	returns attr. name
`rsmd.getColumnDisplaySize()`	returns width

A final 'FYI': To get a result's cardinality, call in sequence:

`answer.last()`	moves to last tuple
`answer.getRow()`	to get current row number

Announcements

- Homework #3 (SQL Queries) is due now.
  - Can use 1 late day, if needed and you still have one remaining.
- Program #3 (JDBC) is due a week from Wednesday (that is, due Nov. 2nd).
- Grading:
  - Exam #1 is on-going
  - Homework #2's grading will start soon
- Reminder: The university moved the withdrawal deadline to the last day of classes.

Creating Indices (1 / 3)

Form:

CREATE [UNIQUE] **INDEX** <*index name*>
ON <*table name*>
[USING <*access method*>]
(<*attribute name*> [, <*attribute name*> . . .]);

type of index!

Creating Indices (2 / 3)

Example(s):

Create an index on `jno` in SPJ:

```
create index spj_j_index
on spj (jno);
```

Remember: DBMSes usually create a unique index on primary keys, automatically.

Creating Indices (3 / 3)

Different DBMSes supply different kinds of indices; e.g.:

1. Oracle 11:
   - B–tree
     - Reverse Key (subtype of B–Tree, reverses bytes)
   - Function–based (to support queries using computations)
   - Bitmap (instead of storing lists of IDs)
   - Application Domain Indexes (user–defined)

2. Postgres 14:
   - B–tree
   - Hash (apparently linear hashing)
   - GiST (Generalized Search Tree) and SP–GiST
   - GIN (Generalized Inverted Index)
   - BRIN (Block Range Index)

Creating Views (1 / 2)

Remember the ANSI/SPARC External Layer?

Form:

CREATE VIEW <*view name*> [(<*attribute list*>)]

AS <*select statement*>;

- once created, use the view as a relation
- we can create views "on the fly" (as needed) or once but store it. (materialized view)

Creating Views (2 / 2)

Example(s):

Create a view of supplier names and the IDs of the parts that they supply.

```
create view supplierpart ("Supplier Name", "Part #")
as   select distinct sname, pno
     from    s, spj
     where   s.sno = spj.sno;
```

Then, it is available for use:

```
select * from supplierpart;
```

View Updates (1 / 2)

Can users update the content of views? That is, can we convert a view update into updates of the view's base relations?

Example(s):

Consider a view that is a join of A and B:

A

a	b	c
x	2	b
y	1	a
z	1	b

B

d	a
6	y
1	y

A ⋈ B

a	b	c	d
y	1	a	6
y	1	a	1

Can we insert the tuples [y 1 a 4] and [x 2 c 3] into this view?

View Updates (2 / 2)

Example(s): (continued!) Our desired result:

$A \bowtie B$

a	b	c	d
y	1	a	6
y	1	a	1
y	1	a	4
x	2	c	3

$\Rightarrow$

A

a	b	c
x	2	b
y	1	a
z	1	b
x	2	c

B

d	a
6	y
1	y
4	y
3	x

Problems

- inserting [y 1 a 4] ... just have to note that y 1 a is already there.
- inserting [x 2 c 3] ...
  (1) adds another 'x' value to a attribute!
  (2) causes [x 2 b 3] to appear from A ⋈ B
  - that's a spurious tuple.

⇒ Many DBMSs do not allow view updates.

SQL as DML

The view update example raises a pertinent question:

How *do* we insert data into a relation?

With a DML operation, of course!

Inserting Tuples into a Relation

Review

To insert a tuple into a relation:

INSERT INTO <*relation name*> [(<*column list*>)]

VALUES (<*expression list*>);

Example(s):

```
insert into foo values ('z', 1, 'b');
```

Notes:

- assumes values are listed in the relation's defined attribute order
- best to disable transactions (set autocommit off) if doing many INSERT INTOs in sequence.

Bulk Loading a Database

Using INSERT INTO to populate tables is:

- Highly portable! (just create a script file), but
- Slow (especially you don't disable transations)

An alternative is a bulk–loading utility.

Example(s):

Oracle: sqlldr utility program

Postgres: COPY command

Updating Content of Tuples

To modify data in existing tuples:

UPDATE <*relation name*>

SET <*attribute name*> = <*expression*> [, . . .]

[FROM <*relation list*>]

[WHERE <*condition*>];

Example(s):

```
update m
set name = 'Ray'
where id = 1;
```

Storing Query Results

Can we add query results (which are relations) to the DB?

Yes! Two options:

1. (Pretty universal) If you have an existing table:

   INSERT INTO <*relation name*>

   <*SELECT statement*>;

2. (Oracle) If you need to create the table, too:

   CREATE GLOBAL TEMPORARY TABLE <*relation name*>

   AS <*SELECT statement*>;

   (Table disappears at end of session.)

Deleting Tuples

Like updating, a condition is used to ID tuples for removal:

DELETE FROM <*relation name*>

WHERE <*condition*>;

What if we don't use the WHERE clause?

Eg: Delete from S;

All tuples will be deleted, but not the relation.

Deleting Relations

To remove tables, indices, views, . . .

DROP { TABLE | INDEX | VIEW | DATABASE } <*name*>;

Wait! What About “SQL as DCL?”

We’ll cover that in Topic 14: Security.

Topic 10:

Transactions and Assertions

What is a Transaction?

The situation:

Individual SQL statements are often pieces of multi–step actions that a DBMS must manage.

Definition: Transaction

A ("xact") is a group of one or more operations treated as a single, indivisible unit of work.

Critical for concurrency control, crash recovery

The ACID Properties of Transactions

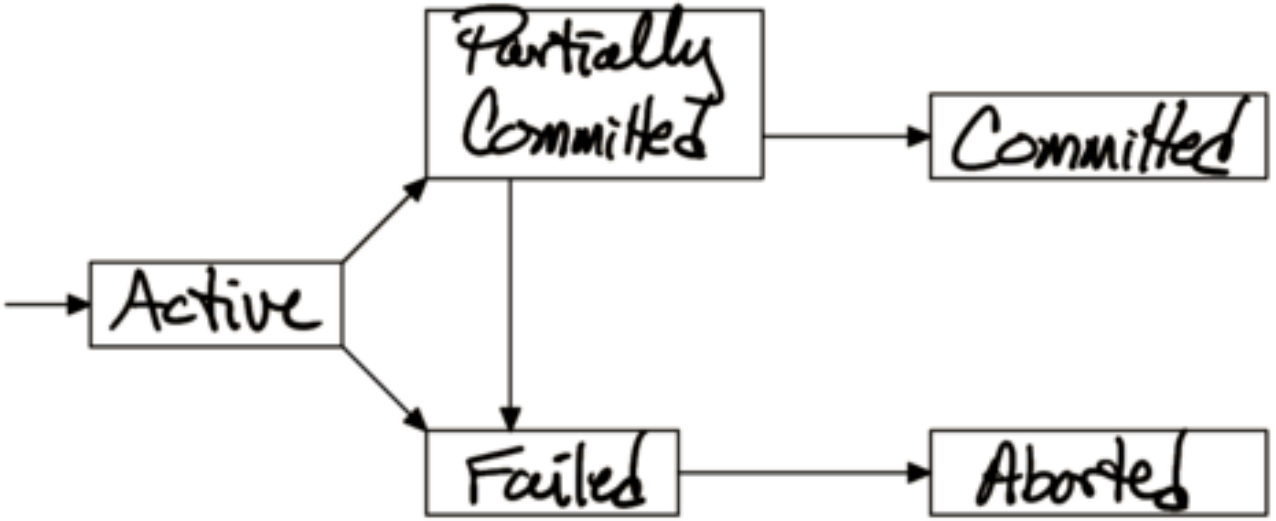
A is for Atomicity: Xacts are all or nothing.

C is for Consistency: An xact's actions retain the validity of the DB's content.

I is for Isolation: Each xact appears to be the only xact running in the DBMS.

D is for Durability: A completed xact's changes are permanent, regardless of crashes, disk failures, etc.

Transaction Lifetime (1 / 2)

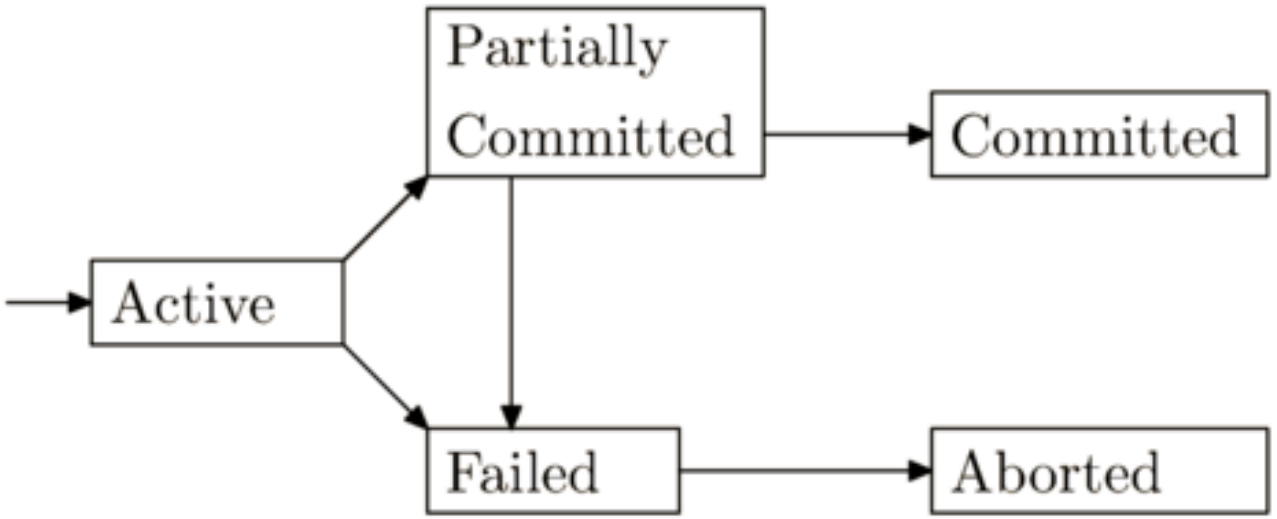


Active: The xact is being executed.

Failed: The xact did not finish, but the partial changes still exist in the DB. Need to reset the DB.

Aborted: The xact appears to have never started.

Transaction Lifetime (2 / 2)



Partially Committed – Xact's actions have completed, but changes may not yet be on non-volatile storage.

Committed – Changes are safely on disk.

Announcements

- Program #3 is due a week from today (that is, November 2nd).
- Exam #1 is still on-track to be graded by the end of the week.
  - After which, the TAs will get to work on grading homework.
  - I'll update the score page w/ Ex 1 when it has been graded.
- Looking ahead:
  - Next assignment is Homework 4
  - Exam #2 is Nov 16th (3 weeks)

Transaction Isolation

Observation: In Oracle's PL/SQL, every action is automatically part of a transaction.

To stop a transaction (and start a new one), either:

(1) commit; ← retain effects

(2) rollback; ← undo effects

To make each PL/SQL statement its own transaction:

set autocommit on; (overhead)

Transaction Isolation Demo (1 / 2)

Each 'user' is an Oracle login in a separate terminal window:

	User 1	User 2
(1)	`@ xact.sql`	—
(2)	`show autocommit;` ⇒ "autocommit OFF"	—
(3)	`select * from score;`	—
(4)	—	`select * from score;` ⇒ no rows are selected

(5)	—	`select table_name`
		`from user_tables;`
		⇒ yes, score exists!

(6)	`commit;`	—
(7)	—	`select * from score;` ⇒ score's content is visible

(Continues . . .)

Transaction Isolation Demo (2 / 2)

	User 1	User 2
(8)	`insert into score values (5,460,'B');`	—
(9)	`select * from score;` ⇒ shows it	—
(10)	—	`select * from score;` ⇒ doesn't show it
(11)	`rollback;`	—
(12)	`select * from score;` ⇒ like it was never there	—
(13)	`set autocommit on;`	—
(14)	`insert into score values (4,453,'B');`	—
(15)	—	`select * from score;` ⇒ shows it

Constraints in SQL

Consider:

```
create table applicant (
   id     integer,
   email  char(30) not null,
   ...
   primary key (id)
);
```

Assertions (1 / 2)

The SQL standard provides for general assertions.

Example(s): No one in 460 can receive an 'E':

```
create assertion no_460_Es
check (not exists (select *
                     from score
                    where course = 460
                      and grade = 'E') );
```

Trying to add a grade of 'E' will fail.

Sadly, not supported in Oracle.

Assertions (2 / 2)

. . . Oracle supports a form of general constraint within 'create table':

Example(s): No one in 460 can receive an 'E':

```
create table score (
  ...
  constraint no_fail check (grade <> 'E')
);
```

if the condition is True, insertion is allowed,

insert into score values (5, 453, 'E');
⇒ fails.

Trigger Basics (1 / 2)

- Triggers support the idea of 'active databases' (events initiate predetermined actions)
- Oracle does support these (stay tuned!)
- Triggers follow the "ECA" model:
  - Event - Causes trigger activation (eg. DML Command)
  - Condition - if true, action is performed
  - Action - statements to be executed.
- Useful for input validation and update logging tasks

Trigger Basics (2 / 2)

Some Disadvantages of Triggers:

1. Hard to write the appropriate actions
   - Both for now and the future.
2. Specified separately from relations(s)
   - easy to forget about
3. Can reduce the DBMS' concurrency
   - triggers can become a bottleneck
4. Generally hard to anticipate how the triggers will interact
   - e.g. rules could trigger each other indefinitely.

Triggers in Oracle (1 / 4)

Oracle's basic trigger definition syntax:

```
create trigger <name>
{before/after} {insert/delete/update of <attr>} on <relation>
[ [ for each row ] when ( < condition > ) ]
< PL/SQL block > ;
```

Component meanings:

- "**`for each row`**" gives row–level triggers (vs. statement–level):
  - "row–level": trigger executes when a row is changed
    - '**`before`**' – fires before a new value is written
    - '**`after`**' – fires after value is written; good for validation
  - "stmt–level": trigger executed when SQL statement is executed
- The PL/SQL block can be a compound statement
- Only use triggers when necessary – execution order not guaranteed!

--- unless you use FOLLOWS in Oracle (added 2017)

Triggers in Oracle (2 / 4)

Oracle's Create Trigger command does only that — creates.

To activate the trigger, follow it with either:

(a) `.` ← (period) terminates subprogram creation

`run;` ← execute PL/SQL subprogram

(b) `/` ← (slash) merges [.] and [run;]

Triggers in Oracle (3 / 4)

We want to know if someone tries to add a 460 'E' in score:

Example(s):

```
create trigger no_460_Es
after insert on score
for each row
when ( new.course = 460 and new.grade = 'E' )
 begin
  raise_application_error (-20000, 'message');
 end no_460_Es;
/
```

Triggers in Oracle (4 / 4)

Notes:

1. Could we use a trigger to change an inserted 'E' to a 'D'?
   - No. We can't change the table that triggered the rule currently being executed. Oracle will report a "mutating table" error.
2. It's easy to create syntax errors when writing triggers
   - Use `sho err` to see the last compilation error
3. Removing a trigger is easy
   - Use `drop trigger <name>;`

Topic 11:

Graphical Query Languages

QBE (Query By Example)

- Mid-1970s creation of Moshe Zloof at IBM Research
- Associated with development of Domain Relational Calculus
- Still available within IBM's QMF (Query Management Facility) component of DB2
- Most DBMSes offer a QBE-inspired interface
- Idea: Let users create queries by showing what they want to see

QBE Templates and π

A QBE template is the schema ('skeleton') of a table.

Example(s): The 'table skeleton' of the SPJ table:

SPJ	sno	pno	jno	qty

π: Easy! Type `P.` beneath each field to be displayed.

Example(s): In which cities are the suppliers located?

S	sno	sname	status	city
				P.

QBE and σ

To select tuples, place partial conditions under the attributes.

Example(s):

What are the names and numbers of the active suppliers in London?

S	sno	sname	status	city
	P.	P.	> 0	London

So: No equals sign needed
Can negate with ¬ (e.g. ¬ London)

QBE and ⋈

To force values to match, we add variables

– In QBE, variables begin with an underscore (‘ _ ’).

Example(s):

What are the names of the parts that can be supplied by individual suppliers in quantity > 200?

P	pno	pname	color	weight	city
	_p	P.			

SPJ	sno	pno	jno	qty
		_p		> 200