HW #5 Released Today

Wed. Apr 27th. 4:30-5:30
SC 1302.

Wed Marina: Recovery / Finish.

Fri. Selected Demos

Wed: SP. on NoSQL DBMS
Sumary & Beyond

$A \bowtie B \approx 0.01\,|A|\,|B|$ Sells.beer = Beers.name

Estimating Sizes

Sells $\bowtie_{beer}$ Beers

Estimating the size of a natural join, $R \bowtie_A S$

T(Sells)

- When the set of A values are disjoint, then $T(R \bowtie_A S) = 0$
- When A is a key in S and a foreign key in R, then $T(R \bowtie_A S) = T(R)$

Foreign key

Sells.beer Beers.name

- When A has a unique value, the same in R and S, then $T(R \bowtie_A S) = \min(T(R), T(S))$

bud

bud light

X s.adam

bud

bud light

s.adam

Students

NetID name dept

TAs

NetID course

Students.NetID $\supseteq$ TAs.netID

T(TAs)

Estimating Sizes

Assumptions:

- *Containment of values*: if V(R,A) <= V(S,A), then the set of A values of R is included in the set of A values of S
  - Note: this indeed holds when A is a foreign key in R, and a key in S
- *Preservation of values*: for any other attribute B, $V(R \bowtie_A S, B) = V(R, B)$ (or $V(S, B)$)

$R \bowtie_A S$

$? \; V(R \bowtie S, A)$

$V(\text{Sells.beer}) \leq V(\text{Beers.name})$

$\Downarrow$

$\text{Sells.beer} \subseteq \text{Beers.name}$

"referential integrity"

$V(\text{students} \bowtie_{\text{netID}} \text{TAs}, \text{dept})$

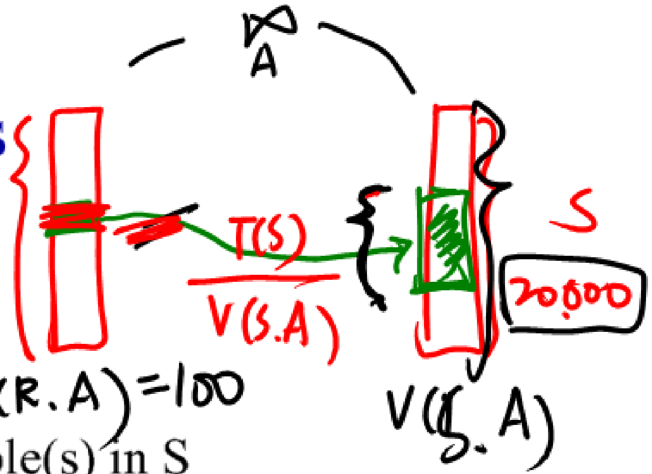
$= V(\text{Student}, \text{Dept}) = 3 \; \{\text{"CS"}, \text{"ECE"}, \text{"ME"}\}$

$T(R) \times \dfrac{T(S)}{V(S.A)}$

Estimating Sizes

Containment Assumption

$10000 \times \dfrac{20.000}{200} = 1000000$ R 10,000



Assume V(R,A) <= V(S,A)

- Then each tuple t in R joins *some* tuple(s) in S
  - How many ?
  - On average S/V(S,A)
  - t will contribute S/V(S,A) tuples in R $\bowtie_A$ S
- Hence T(R $\bowtie_A$ S) = T(R) T(S) / V(S,A)

Sells_beer 100

Beers.name = 200

200

In general: T(R $\bowtie_A$ S) = $\boxed{\text{T(R) T(S) / max(V(R,A),V(S,A))}}$

$\dfrac{20000}{200} = 100$?

bud
bud
:

budwiser
Miller

Now. name is not key

Estimating Sizes

Example:

- $T(R) = 10000$, $T(S) = 20000$
- $V(R,A) = 100$, $V(S,A) = 200$
- How large is $R \bowtie_A S$?

Answer: $T(R \bowtie_A S) = 10000 * 20000/200 = 1M$

Estimating Sizes

Joins on more than one attribute:

- $T(R \bowtie_{A,B} S) =$

  $T(R)\ T(S)/\max(V(R,A),V(S,A))\max(V(R,B),V(S,B))$

Histograms

- Statistics on data maintained by the RDBMS
- Makes size estimation much more accurate (hence, cost estimations are more accurate)

T(R)

Histograms

Employee(<u>ssn</u>, name, salary, phone)

- Maintain a histogram on salary:

Salary:	0..20k	20k..40k	40k..60k	60k..80k	80k..100k	> 100k
Tuples	200	800	5000	12000	6500	500

- T(Employee) = 25000, but now we know the distribution

Histograms

Ranks(rankName, salary)

- Estimate the size of Employee $\bowtie_{\text{Salary}}$ Ranks

Employee	0..20k	20k..40k	40k..60k	60k..80k	80k..100k	> 100k
	200	800	5000	12000	6500	500

Ranks	0..20k	20k..40k	40k..60k	60k..80k	80k..100k	> 100k
	8	20	40	80	100	2

Histograms

- Assume:
  - V(Employee, Salary) = 200
  - V(Ranks, Salary) = 250
- Then T(Employee $\bowtie_{Salary}$ Ranks) =

  $= \Sigma_{i=1,6} T_i T_i' / 250$

  $= (200x8 + 800x20 + 5000x40 +$
  $12000x80 + 6500x100 + 500x2)/250$

  $= ....$

CS411
Database Systems

Most Novel Concept
DB has contributed C.S.

10: Transaction Management

NetID ______ Name ______

Q1: Name one thing this class must change.

Q2: Name one thing this class should keep.

Why Do We Learn This?

"Programmer"

Database Program = Transaction

$\neq$ SQL stmt.

Outline

- Transaction management
  - motivation & brief introduction
  - major issues
    - recovery
    - concurrency control
- Recovery

Users and DB Programs

- End users don't see the DB directly
  - are only vaguely aware of its design
  - may be acutely aware of part of its contents
  - SQL is not a suitable end-user interface
- A single SQL query is not a sufficient unit of DB work
  - May need more than one query
  - May need to check constraints not enforced by the DBMS
  - May need to do calculations, realize "business rules", etc.

Users and DB Programs

- Ergo, a program is needed to carry out each unit of DB work
- End users interact with DB programs
  - Rather than SQL
  - May be many users simultaneously
    - Thus many simultaneous executions of these programs
  - Each user expects service and correct operation
    - A user should not have to wait forever
    - A user should not be affected by errors of others

why interleave ?

- Fairness
- Utilization

Definition of "Transaction"

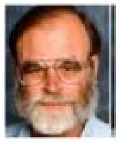
Definition: A transaction is the execution of a DB program.

- DB applications are designed as a set of transactions
- Typical transaction = Prog
  - starts with data from user or from another transaction
  - includes DB reads/writes
  - ends with display of data or form, or with request to start another transaction

Behind the Scene: Who invented Transaction?

- Edgar Codd?
- Jim Gray?
- Al Gore?

1.1 Historical Perspective



Six thousand years ago, the Sumerians invented writing for transaction processing. The earliest known writing is found on clay tablets recording the royal inventory of taxes, land, grain, cattle, slaves, and gold; scribes evidently kept records of each transaction. This early system had the key aspects of a transaction processing system (see Figure 1.1):

Database. An abstract system state, represented as marks on clay tablets, was maintained. Today, we would call this the *database*.

Transactions. Scribes recorded state changes with new records (clay tablets) in the database. Today, we would call these state changes *transactions*.

The Sumerians' approach allowed the scribes to easily ask questions about the current and past state, while providing a historical record of how the system got to the present state.

The technology of clay-based transaction processing systems evolved over several thousand years through papyrus, parchment, and then paper. For over a thousand years, pa-

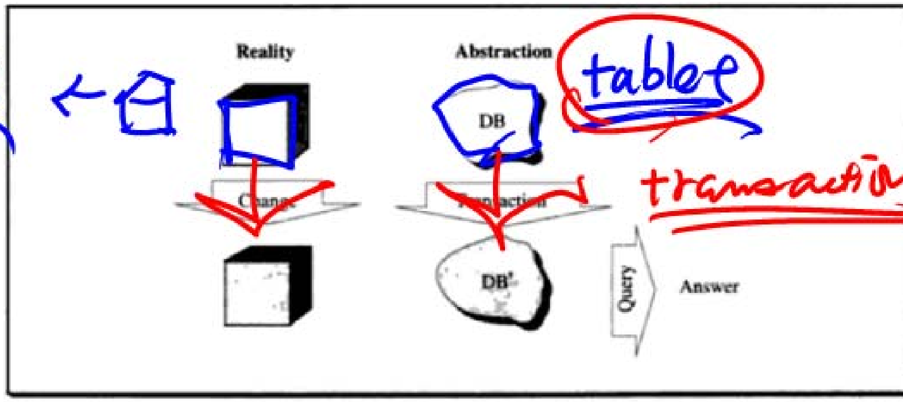


Figure 1.1: The basic abstraction of transaction processing systems. The real state is represented by an abstraction, called the *database*, and the transformation of the real state is mirrored by the execution of a program, called a *transaction*, that transforms the database

And He is a Key Contributor

Dr. Jim Gray worked at the IBM San Jose Research Laboratory from October 1972 until September 1980. During that time he developed and implemented the foundational techniques that underlie and enable on-line transaction processing. The deployment of on-line transaction processing reduces the cost of business transactions by reducing delays and eliminating paper records. Dr. Gray received the 1998 A.M. Turing Award *"For fundamental contributions to database and transaction processing research and technical leadership in system implementation from research prototypes to commercial products. The transaction is the fundamental abstraction underlying database system concurrency and failure recovery. Gray's work [defined] the key transaction properties: atomicity, consistency, isolation and durability, and his locking and recovery work demonstrated how to build ... systems that exhibit these properties."*

Sys R

#3

From: Jim Gray at IBM: the transaction processing revolution. Bruce G. Lindsay. ACM SIGMOD Record. 37(2). June 2008.

Atomicity

- Transactions must be "atomic"
  - Their effect is all or none
  - DB must be consistent before and after the transaction executes (not necessarily during!)
- EITHER
  - a transaction executes fully and "commits" to all the changes it makes to the DB
  - OR it must be as though that transaction never executed at all

A Typical Transaction

- ***User view: "Transfer money from savings to checking"***
- Program:
  - read savings;
  - verify balance is adequate;
  - update savings balance;
  - read checking;
  - update checking balance;

"Commit" and "Abort"

- A transactions which only READs expects DB to be consistent, and cannot cause it to become otherwise.
- When a transaction which does any WRITE finishes, it must either
  - COMMIT: "I'm done and the DB is consistent again" OR
  - ABORT (ROLLBACK): "I'm done but I goofed: my changes must be undone."

ATM

Run away

dispense $S cash

if bal ≥ S

bal = bal − S

else

rollback

Complications

- A DB may have many simultaneous users
  - simultaneous users implies simultaneous transactions implies simultaneous DB access
    - multiprogramming/multiprocessing
- Things can go wrong!
  - transactions can conflict with one another
  - programs may crash, OS may crash, disk may crash
    - company loses customer, gets sued, goes bankrupt, etc.

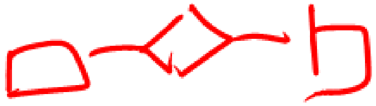
But DB Must Not Crash

- Can't be allowed to become inconsistent
  - A DB that's 1% inaccurate is 100% unusable.
- Can't lose data
- Can't become unavailable

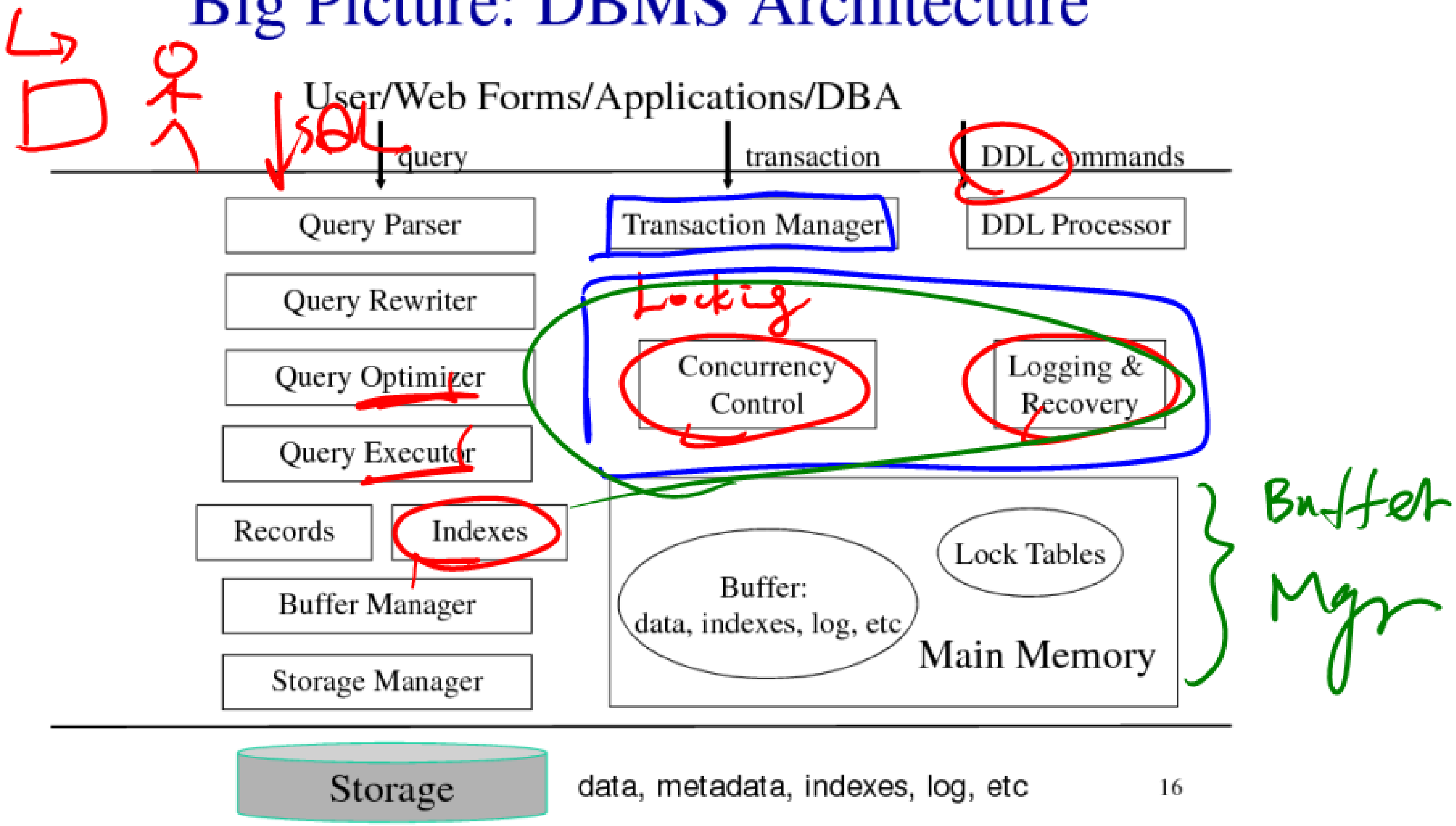
Can you name information processing systems that are more error tolerant?

Transaction Manager (or TP Monitor)

- Part of the DBMS
- Main duties:
  - Starts transactions
    - locate and start the right program
    - ensure timely, fair scheduling
  - Logs their activities
    - especially start/stop, writes, commits, aborts
  - Detects or avoids conflicts
  - Takes recovery actions



Big Picture: DBMS Architecture



What's on the Log File?

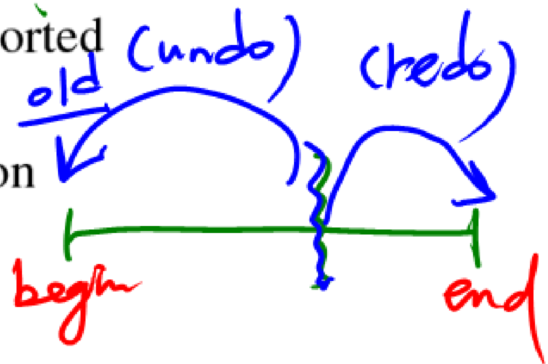
= Journal

- Transaction starts/stops
- DB writes: "before" and/or "after" images of DB records

  old new

  – befores can be used to rollback an aborted transaction

  – afters can be used to redo a transaction (recovery from catastrophe)



- COMMITs and ABORTs

The log itself is as critical as the DB!

The Big TP Issues

- Recovery
  - Taking action to restore the DB to a consistent state
- Concurrency Control
  - Making sure simultaneous transactions don't interfere with one another

The ACID Properties

- Atomicity
- Consistency Preservation
- Isolation
- Durability

The ACID Properties: From Oracle Wiki

ACID

ACID refers to the basic properties of a database transaction: **A**tomicity, **C**onsistency, **I**solation, and **D**urability.

All Oracle database, Oracle RDB and InnoDB transactions comply with these properties. However, Oracle's Berkeley DB database is not ACID-compliant.

Atomicity

The entire sequence of actions must be either completed or aborted. The transaction cannot be partially successful.

Consistency

The transaction takes the resources from one consistent state to another.

Isolation

A transaction's effect is not visible to other transactions until the transaction is committed.

Durability

Changes made by the committed transaction are permanent and must survive system failure.

Behind the Scene: It's Your Turn!

So, who coined "ACID"?

Al Gore ?

Bush x ?

Jim Gray ?

Sumerians x

Recovery

Basement

Q: What Might Go Wrong?

System Failures

- Each transaction has *internal state*
- When system crashes, internal state is lost
  - Don't know which parts executed and which didn't
- Remedy: use a <u>**log**</u>
  - A file that records every single action of the transaction