PART II

• Crash recovery (2 lectures) Ch.17
• Concurrency control (3 lectures) Ch.18
• Transaction processing (2 lectures) Ch.19
• Information integration (1 lect) Ch.20

Integrity or correctness of data

• Would like data to be “accurate” or “correct” at all times

EMP

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>White</td>
<td>52</td>
</tr>
<tr>
<td>Green</td>
<td>3421</td>
</tr>
<tr>
<td>Gray</td>
<td>1</td>
</tr>
</tbody>
</table>

Integrity or consistency constraints

• Predicates data must satisfy
• Examples:
  - x is key of relation R
  - x → y holds in R
  - Domain(x) = {Red, Blue, Green}
  - α is valid index for attribute x of R
  - no employee should make more than twice the average salary

Definition:

• Consistent state: satisfies all constraints
• Consistent DB: DB in consistent state

Constraints (as we use here) may not capture “full correctness”

Example 1  Transaction constraints

• When salary is updated, new salary > old salary
• When account record is deleted, balance = 0
Note: could be “emulated” by simple constraints, e.g.,

| account | Acct # | … | balance | deleted |

Constraints (as we use here) may not capture “full correctness”

Example 2: Database should reflect real world

Example: $a_1 + a_2 + \ldots + a_n = TOT$ (constraint)

Deposit $100 in a_2$: $a_2 \leftarrow a_2 + 100$

$TOT \leftarrow TOT + 100$

Transaction: collection of actions that preserve consistency

Consistent DB $\rightarrow T \rightarrow$ Consistent DB

Big assumption:

If T starts with consistent state + T executes in isolation

$\Rightarrow$ T leaves consistent state
Correctness (informally)
- If we stop running transactions, DB left consistent
- Each transaction sees a consistent DB

How can constraints be violated?
- Transaction bug
- DBMS bug
- Hardware failure
  - e.g., disk crash alters balance of account
- Data sharing
  - e.g.: T1: give 10% raise to programmers
  - T2: change programmers ⇒ systems analysts

How can we prevent/fix violations?
- Chapter 17: due to failures only
- Chapter 18: due to data sharing only
- Chapter 19: due to failures and sharing

Will not consider:
- How to write correct transactions
- How to write correct DBMS
- Constraint checking & repair
  That is, solutions studied here do not need to know constraints

Chapter 17 Recovery
- First order of business:
  Failure Model

Events — Desired
  Undesired — Expected
  Unexpected
Our failure model

**Processor**

**Memory** → **CPU**

**Disk**

Desired events: see product manuals…

Undesired expected events:
- System crash
- Memory lost
- CPU halts, resets

Undesired Unexpected: Everything else!

Examples:
- Disk data is lost
- Memory lost without CPU halt
- CPU implodes wiping out universe…

Undesired Unexpected: Everything else!

Is this model reasonable?

Approach: Add low level checks + redundancy to increase probability model holds

E.g.,
- Replicate disk storage (stable store)
- Memory parity
- CPU checks

Second order of business:

Storage hierarchy

Operations:
- **Input (x)**: block with x → memory
- **Output (x)**: block with x → disk
- **Read (x,t)**: do input(x) if necessary t ← value of x in block
- **Write (x,t)**: do input(x) if necessary value of x in block ← t
**Key problem**  Unfinished transaction

**Example**  Constraint: $A = B$

$T_1$: $A \leftarrow A \times 2$

$B \leftarrow B \times 2$

---

$T_1$:  
Read $(A, t)$; $t \leftarrow t \times 2$
Write $(A, t)$;
Read $(B, t)$; $t \leftarrow t \times 2$
Write $(B, t)$;
Output $(A)$;
Output $(B)$;  

failure!

$A: 16$

$B: 16$

memory

---

$A: 8$

$B: 8$

disk

---

• **Need** atomicity: execute all actions of a transaction or none at all

---

**One solution:** undo logging (immediate modification)

due to: Hansel and Gretel, 782 AD

• Improved in 784 AD to durable undo logging

---

**Undo logging** (Immediate modification)

$T_1$:  
Read $(A, t)$; $t \leftarrow t \times 2$
Write $(A, t)$;
Read $(B, t)$; $t \leftarrow t \times 2$
Write $(B, t)$;
Output $(A)$;
Output $(B)$;

$A: 16$

$B: 16$

memory

---

$A: 16$

$B: 16$

log

---

**One "complication"**

• Log is first written in memory
• Not written to disk on every action

---

$A: 16$

$B: 8$

DB

BAD STATE # 1
One “complication”
• Log is first written in memory
• Not written to disk on every action

Undo logging rules
(1) For every action generate undo log record (containing old value)
(2) Before x is modified on disk, log records pertaining to x must be on disk (write ahead logging: WAL)
(3) Before commit is flushed to log, all writes of transaction must be reflected on disk

Recovery rules: Undo logging
(1) Let S = set of transactions with <Ti, start> in log, but no <Ti, commit> (or <Ti, abort>) record in log
(2) For each <Ti, X, v> in log, in reverse order (latest → earliest) do:
   - if Ti ∈ S then
     - write (X, v)
     - output (X)
   - write <Ti, abort> to log

Recovery rules: Undo logging
(1) Let S = set of transactions with <Ti, start> in log, but no <Ti, commit> (or <Ti, abort>) record in log
(2) For each <Ti, X, v> in log, in reverse order (latest → earliest) do:
   - if Ti ∈ S then
     - write (X, v)
     - output (X)
   - write <Ti, abort> to log

What if failure during recovery?
No problem! / Undo idempotent

To discuss:
• Redo logging
• Undo/redo logging, why both?
• Real world actions
• Checkpoints
• Media failures
Redo logging (deferred modification)

T1: Read(A,t); t t=2; write (A,t);
     Read(B,t); t t=2; write (B,t);
     Output(A); Output(B)

Redo logging rules

1. For every action, generate redo log record (containing new value)
2. Before X is modified on disk (DB), all log records for transaction that modified X (including commit) must be on disk
3. Flush log at commit

Recovery rules:

- For every Ti with <Ti, commit> in log:
  - For all <Ti, X, v> in log:
    - Write(X, v)
    - Output(X)

Solution: Checkpoint (simple version)

Periodically:
1. Do not accept new transactions
2. Wait until all transactions finish
3. Flush all log records to disk (log)
4. Flush all buffers to disk (DB) (do not discard buffers)
5. Write “checkpoint” record on disk (log)
6. Resume transaction processing
Example: what to do at recovery?

Redo log (disk):

```
... <T1,16> ...<T1,commit> ...Checkpoint ...
<T2,A17> ...<T2,commit> ...<T3,C21> ...
```

Crash...

Key drawbacks:
- **Undo logging**: cannot bring backup DB copies up to date
- **Redo logging**: need to keep all modified blocks in memory until commit

Solution: undo/redo logging!

Update \(\Rightarrow\) \(<Ti, Xid, New X val, Old X val>\)
page X

Rules
- Page X can be flushed before or after Ti commit
- Log record flushed before corresponding updated page (WAL)
- Flush at commit (log only)

Non-quiesce checkpoint

Start-ckpt active TR: Ti,T2,...
end ckpt ...

LOG ...
... dirty buffer pool pages flushed ...

for undo ...

Examples what to do at recovery time?

```
LOG ...
... Ti,a ... Ckpt T1 ... Ckpt end ...
T1- b ...
```

\(\Rightarrow\) Undo T1 (undo a,b)
Example

\[
\begin{array}{c}
L \quad O \quad G \\
\ldots T_1 \quad a \quad \ldots \quad kpt-s \quad T_1 \quad b \quad \ldots \quad kpt-end \quad T_1 \quad c \quad \ldots \quad T_1 \quad cmt \quad \ldots
\end{array}
\]

\(\text{Redo } T_1: (\text{redo } b,c)\)

Recovery process:
- Backwards pass (end of log \(\leftarrow\) latest checkpoint start)
  - construct set \(S\) of committed transactions
  - undo actions of transactions not in \(S\)
- Undo pending transactions
  - follow undo chains for transactions in (checkpoint active list) \(\leftarrow S\)
- Forward pass (latest checkpoint start \(\leftarrow\) end of log)
  - redo actions of \(S\) transactions

Real world actions
E.g., dispense cash at ATM
\(T_i = a_1 \quad a_2 \ldots \quad a_j \ldots \quad a_n\)

\(\$\)

Solution
(1) execute real-world actions after commit
(2) try to make idempotent

Media failure (loss of non-volatile storage)
A: 16

Solution: Make copies of data!
Example 1  Triple modular redundancy
• Keep 3 copies on separate disks
• Output(X) --> three outputs
• Input(X) --> three inputs + vote

Example #2  Redundant writes, Single reads
• Keep N copies on separate disks
• Output(X) --> N outputs
• Input(X) --> Input one copy
  - if ok, done
  - else try another one

Assumes bad data can be detected

Example #3: DB Dump + Log

• If active database is lost,
  - restore active database from backup
  - bring up-to-date using redo entries in log

When can log be discarded?

check-point

log
db
dump
redo
needed
not
needed
undo
media
recovery
undo
after
system
failure
undo
after
system
failure

Summary
• Consistency of data
• One source of problems: failures
  - Logging
  - Redundancy
• Another source of problems: Data Sharing..... next