CS4432: Database Systems II

Lecture #19

Database Consistency and Violations?

Professor Elke A. Rundensteiner
Transactions, etc.

• Crash recovery
  Ch.8 [17]
• Concurrency control
  Ch.9 [18]
• Transaction processing
  Ch.10 [19]
All about Project 3.
Integrity or correctness of data?

• We would like data in our database to be “accurate” ( “correct” ) at all times.

<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>

• How DBMS decides if data is consistent?
Integrity or consistency constraints

- Utilize **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
Definitions:

- **Consistent state**: satisfies all constraints
- **Consistent DB**: DB in consistent state
Such Constraints may not capture “full correctness”

Example 1: Transaction constraints

- When salary is updated, new salary > old salary
- When account record is deleted, balance = 0
Constraints (as we use here) may not capture “full correctness”

Example 2: Database should reflect real world
in any case, continue with constraints...

Observation: DB cannot be consistent always

Example:

Constraint: \( a_1 + a_2 + \ldots + a_n = TOT \)

Action:

Deposit $100 in \( a_2 \):

\[
\begin{align*}
\text{a}_2 & \leftarrow \text{a}_2 + 100 \\
\text{TOT} & \leftarrow \text{TOT} + 100
\end{align*}
\]
Example: \( a_1 + a_2 + \ldots + a_n = \text{TOT} \) (constraint)

Deposit $100 in \( a_2 \): \[ a_2 \leftarrow a_2 + 100 \]
\[ \text{TOT} \leftarrow \text{TOT} + 100 \]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>50</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{TOT} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td></td>
</tr>
</tbody>
</table>

\[ \rightarrow \]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>150</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{TOT} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td></td>
</tr>
</tbody>
</table>

\[ \rightarrow \]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>150</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( \text{TOT} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1100</td>
<td></td>
</tr>
</tbody>
</table>
Transaction: a collection of actions that preserve consistency
Big assumption:

If $T$ starts with \textit{consistent state}  
\textbf{AND}  
$T$ executes in \textit{isolation}  
\Rightarrow $T$ leaves consistent state
Correctness  (informally)

- If we stop running transaction(s), 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
Will not consider:

- How to write correct transactions
- How to write correct DBMS system
- Constraint checking & repair
How can we prevent/fix violations?

- Chapter 8[17]: due to failures only
- Chapter 9[18]: due to data sharing only
- Chapter 10[19]: due to failures and sharing
Chapter 8 [17]: Recovery

• First: Failure Model
Events — Desired
Undesired — Expected
Unexpected
Our failure model

CPU → processor

memory → M

disk ← D

CS4432 recovery 19
Desired events: see product manuals....

Undesired expected events:
- System crash
  - memory lost
  - cpu halts, resets

Undesired Unexpected: Everything else!

that’s it!!
Undesired Unexpected: Everything else!

Examples:
• Disk data is lost
• Memory lost without CPU halt
• CPU implodes wiping out universe....
• You name it ...
Is this model reasonable?

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

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