CS 839: Design the Next-Generation Database
Lecture 2: Transaction Basics

Xiangyao Yu
1/23/2020
Announcements

Course website


Email me if you are not in HotCRP

https://wisc-cs839-ngdb20.hotcrp.com
Today’s Agenda

OLTP vs. OLAP

ACID properties

• Atomicity
• Consistency
• Isolation
• Durability
OLTP vs. OLAP

OLTP: On-Line Transaction Processing
• Users submit transactions that contain simple read/write operations
• Example: banking, online shopping, etc.

OLAP: On-Line Analytical Processing
• Complex analytics queries that reveal insights behind data
• Example: business report, marketing, forecasting, etc.
OLTP vs. OLAP

Transactions

OLTP database
(Update Intensive)
OLTP vs. OLAP

Transactions

OLTP database
(Update Intensive)

OLAP database
(Read Intensive, rare updates)
OLTP vs. OLAP

Transactions

OLTP database
(Update Intensive)

- Takes hours for conventional databases
- Takes seconds for Hybrid transactional/analytical processing (HTAP) systems

OLAP database
(Read Intensive, rare updates)
OLTP vs. OLAP

This lecture

OLTP database
(Update Intensive)

Next lecture

OLAP database
(Read Intensive, rare updates)
Transaction Definition

**transaction**

/ˈtrænsækʃən, ˈtrænsækʃən/

*noun*

noun: transaction; plural noun: transactions

an instance of buying or selling something; a business deal.
"in an ordinary commercial transaction a delivery date is essential"

**Similar:** deal, business, agreement, undertaking, affair
Transaction Definition

What are the required properties of a database transaction?

ACID
Transaction Definition

What are the required properties of a database transaction?

**ACID**

Example transactions:

```
Begin
Read(X)
Write(Y)
Insert(Z)
Commit
If checking.balance > 100
   checking.balance -= 100
saving.balance += 100
```
ACID: Atomicity

If checking.balance > 100

checking.balance -= 100

saving.balance += 100

Atomicity: Either all operations occur, or nothing occurs (All or nothing)
ACID: Consistency

If checking.balance > 100
    checking.balance -= 100
    saving.balance += 100

Checking   $50  RunTxn()  -$50
Saving     $1000 $1100

Consistency: Integrity constraints must be maintained.

Example Integrity constraint:
    balance of checking account must be above $0
Example transaction:

If checking.balance > 100
    checking.balance -= 100
    saving.balance += 100

Checking $1000
Saving  $1000
Example transaction:

If checking.balance > 100
    checking.balance -= 100
    saving.balance += 100

<table>
<thead>
<tr>
<th></th>
<th>Checking</th>
<th>Saving</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>Balance</td>
<td>$900</td>
<td>$1100</td>
</tr>
<tr>
<td>RunTxn()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ACID: Durability

Example transaction:

If checking.balance > 100
  checking.balance -= 100
  saving.balance += 100

<table>
<thead>
<tr>
<th></th>
<th>Checking</th>
<th>Saving</th>
</tr>
</thead>
<tbody>
<tr>
<td>Before</td>
<td>$1000</td>
<td>$1000</td>
</tr>
<tr>
<td>After</td>
<td>$900</td>
<td>$1100</td>
</tr>
</tbody>
</table>

RunTxn()  

CRASH !!
**ACID: Durability**

Example transaction:

```plaintext
If checking.balance > 100
    checking.balance -= 100
    saving.balance += 100
```

<table>
<thead>
<tr>
<th></th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checking</td>
<td>$1000</td>
<td>$900</td>
</tr>
<tr>
<td>Saving</td>
<td>$1000</td>
<td>$1100</td>
</tr>
</tbody>
</table>

**Durability**: A transaction’s updates persist in case of system failure.
ACID: Isolation

Initially
checking.balance = 1000
ACID: Isolation

Initially
`checking.balance = 1000`

If `checking.balance > 100`

```
balance = checking.balance
balance = balance - 100
```

`checking.balance = balance`

If `checking.balance > 100`

```
balance = checking.balance
balance = balance - 100
```

`checking.balance = balance`
ACID: Isolation

Initially
\[
\text{checking.balance} = 1000
\]

If \(\text{checking.balance} > 100\)

\[
\text{bal} = \text{checking.balance}
\]

\[
\text{bal} = 1000
\]

\[
\text{bal} = \text{bal} - 100
\]

\[
\text{checking.balance} = \text{bal}
\]

If \(\text{checking.balance} > 100\)

\[
\text{bal} = \text{checking.balance}
\]

\[
\text{bal} = \text{bal} - 100
\]

\[
\text{checking.balance} = \text{bal}
\]
ACID: Isolation

Initially
\[ \text{checking.balance} = 1000 \]

If \( \text{checking.balance} > 100 \)

1. \[ \text{bal} = \text{checking.balance} \]
   \[ \text{bal} = 1000 \]
   \[ \text{bal} = \text{bal} - 100 \]
   \[ \text{checking.balance} = \text{bal} \]

If \( \text{checking.balance} > 100 \)

2. \[ \text{bal} = \text{checking.balance} \]
   \[ \text{bal} = 1000 \]
   \[ \text{bal} = \text{bal} - 100 \]
   \[ \text{checking.balance} = \text{bal} \]
ACID: Isolation

Initially

\[ \text{checking.balance} = 1000 \]

If \( \text{checking.balance} > 100 \)

\[
\begin{align*}
\text{bal} &= \text{checking.balance} \\
\text{bal} &= 1000 \\
\text{bal} &= \text{bal} - 100 \\
\text{bal} &= 900 \\
\text{checking.balance} &= \text{bal}
\end{align*}
\]

If \( \text{checking.balance} > 100 \)

\[
\begin{align*}
\text{bal} &= \text{checking.balance} \\
\text{bal} &= 1000 \\
\text{bal} &= \text{bal} - 100 \\
\text{bal} &= 900 \\
\text{checking.balance} &= \text{bal}
\end{align*}
\]
ACID: Isolation

Initailly
checking.balance = 1000

If checking.balance > 100

1. bal = checking.balance
   bal = 1000
   bal = bal - 100
   bal = 900

2. bal = checking.balance
   bal = 1000
   bal = bal - 100
   bal = 900

3. checking.balance = bal
   checking = 900

If checking.balance > 100

bal = checking.balance
bal = 1000
bal = bal - 100
bal = 900
checking = 900
ACID: Isolation

Initially
checking.balance = 1000

If checking.balance > 100

1. bal = checking.balance
   bal = 1000
   bal = bal - 100
   bal = 900
   checking.balance = bal
   checking = 900

If checking.balance > 100

2. bal = checking.balance
   bal = 1000
   bal = bal - 100
   bal = 900
   checking.balance = bal
   checking = 900
Strong isolation: **Serializability (Focus of this course)**

Schedule of concurrent transactions is equivalent to some serial schedule

Weak isolation: **Snapshot Isolation, Read Committed, Read Uncommitted, etc.**

Weaker isolation levels allow more interleaving of transactions
ACID: Isolation – Why Strong Isolation?

MongoDB & Bitcoin: How NoSQL design flaws brought down two exchanges

DZone April 2014

Attackers stole 896 Bitcoins ≈ 3 million US dollars

Why you should pick strong consistency, whenever possible

Google Cloud January 2018

"Systems that don't provide strong consistency … create a burden for application developers"

SQL (before 2000) -> NoSQL (since 2000) -> NewSQL (since 2010s)
How to Enforce ACID

Atomic & Isolation: Concurrency control

Consistency: Check integrity for transactions

Durability: Logging
Concurrency Control

Pessimistic

Optimistic

½ empty

½ full
Pessimistic – Two Phase Locking (2PL)

T1

Begin

Read(X)

Write(Y)

Commit
Pessimistic – Two Phase Locking (2PL)

T1

Begin

Read(X)

Write(Y)

Commit
Pessimistic – Two Phase Locking (2PL)

T1

Begin

Read(X)  

Write(Y)  

Commit

Time
Pessimistic – Two Phase Locking (2PL)

```
T1
Begin
Read(X)
Write(Y)
Commit

T2
Begin
Write(X)
```
Pessimistic – Two Phase Locking (2PL)

T1

Begin
Read(X)
Write(Y)
Commit

T2

Begin
Write(X)
Pessimistic – Two Phase Locking (2PL)

T1

Begin
Read(X)
Write(Y)
Commit

T2

Begin
Write(X)
Commit

Time
Pessimistic – Deadlock in 2PL

T1

Begin

Read(X)

Write(Y)

T2

Begin

Write(Y)

Write(X)
Pessimistic – Deadlock in 2PL

T1
Begin
Read(X)
Write(Y)

T2
Begin
Write(Y)
Write(X)
Pessimistic – Deadlock in 2PL

T1
Begin
Read(X)
Write(Y)
Write(Y)

T2
Begin
Write(Y)
Write(X)
Pessimistic – Deadlock in 2PL

T1
Begin
Read(X)
Write(Y)

T2
Begin
Write(Y)
Write(X)

Time
Pessimistic – Deadlock in 2PL

T1
Begin
Read(X)
Write(Y)

T2
Begin
Write(Y)
Write(X)
Pessimistic – Deadlock in 2PL

- T1
  - Begin
  - Read(X)
  - Write(Y)

- T2
  - Begin
  - Write(Y)
  - Write(X)

Deadlock!
Pessimistic – Deadlock Resolution

T1
Begin
Read(X)
Write(Y)

T2
Begin
Write(Y)
Write(X)

Detect and break cycles
Allow only certain waits
• NoWait
• WaitDie
• WoundWait
• LimitedDepth
• etc.

Deadlock!
Optimistic Concurrency Control (OCC)

T1

Begin
Read(X)
Write(Y)
Validate(X)
Abort

T2

Begin
Write(X)
Commit
How to Enforce ACID

**Atomic & Isolation**: Concurrency control

**Consistency**: Check integrity for transactions

**Durability**: Logging
Durability: Logging

Initially
checking = 1000

CRASH
Durability: Logging

Initially
checking = 1000

Begin
Write
(Checking = 900)
Logging
Commit

Recovery
Read
(Checking = 900)

Log
(on disk)
T1 T2 T3 T4 T5 ...

CRASH
Summary

A transaction in an OLTP system has ACID properties

Atomicity, Consistency, Isolation, Durability

Concurrency control (enforces Atomicity and Isolation)

Two Phase Locking (2PL)

Optimistic Concurrency Control (OCC)

Logging (enforces Durability)
Discuss the relative advantages and disadvantages of 2PL and OCC.

What is your opinion on the debate between SQL and NoSQL (strong guarantee vs. high performance)?

Do you see any scalability problem with logging? If so, any potential solution?
Before Next Lecture

Submit discussion summary to https://wisc-cs839-ngdb20.hotcrp.com
• One summary per group
• Authors: group members
• Any format is ok (e.g., pdf, doc, txt)
• Feel free to comment on others’ discussion

Submit review for
[optional] C-Store: A Column-oriented DBMS