Transactional Information Systems:

Theory, Algorithms, and the Practice of Concurrency Control and Recovery

Gerhard Weikum and Gottfried Vossen

© 2002 Morgan Kaufmann
ISBN 1-55860-508-8

“Teamwork is essential. It allows you to blame someone else.” (Anonymous)
Part I: Background and Motivation

- 1 What Is It All About?
- 2 Computational Models
Chapter 1: What Is It All About?

- 1.2 Application Examples
- 1.3 System Paradigms
- 1.4 Virtues of Transactions
- 1.5 Architecture of Database Servers
- 1.6 Lessons Learned

“If I had had more time, I could written you a shorter letter”
(Blaise Pascal)
Application Examples

• OLTP, e.g., funds transfer
• E-commerce, e.g., Internet book store
• Workflow, e.g., travel planning & booking
OLTP Example: Debit/Credit

```c
void main ( ) {
    EXEC SQL BEGIN DECLARE SECTION
        int b /*balance*/, a /*accountid*/, amount;
    EXEC SQL END DECLARE SECTION;
    /* read user input */
    scanf ("%d %d", &a, &amount);
    /* read account balance */
    EXEC SQL Select Balance into :b From Account
        Where Account_Id = :a;
    /* add amount (positive for debit, negative for credit) */
    b = b + amount;
    /* write account balance back into database */
    EXEC SQL Update Account
        Set Balance = :b Where Account_Id = :a;
    EXEC SQL Commit Work;
}
```
**OLTP Example 1.1: Concurrent Executions**

<table>
<thead>
<tr>
<th>P1</th>
<th>Time</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select Balance Into :b&lt;sub&gt;1&lt;/sub&gt; From Account Where Account_Id = :a</td>
<td></td>
<td>Select Balance Into :b&lt;sub&gt;2&lt;/sub&gt; From Account Where Account_Id = :a</td>
</tr>
<tr>
<td>/* b&lt;sub&gt;1&lt;/sub&gt;=0, a.Balance=100, b&lt;sub&gt;2&lt;/sub&gt;=0 */</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b&lt;sub&gt;1&lt;/sub&gt; = b&lt;sub&gt;1&lt;/sub&gt;-50</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>/* b&lt;sub&gt;1&lt;/sub&gt;=100, a.Balance=100, b&lt;sub&gt;2&lt;/sub&gt;=100 */</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>/* b&lt;sub&gt;1&lt;/sub&gt;=50, a.Balance=100, b&lt;sub&gt;2&lt;/sub&gt;=100 */</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>/* b&lt;sub&gt;1&lt;/sub&gt;=50, a.Balance=100, b&lt;sub&gt;2&lt;/sub&gt;=200 */</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Update Account Set Balance = :b&lt;sub&gt;1&lt;/sub&gt; Where Account_Id = :a</td>
<td></td>
<td>Update Account Set Balance = :b&lt;sub&gt;2&lt;/sub&gt; Where Account_Id = :a</td>
</tr>
<tr>
<td>/* b&lt;sub&gt;1&lt;/sub&gt;=50, a.Balance=50, b&lt;sub&gt;2&lt;/sub&gt;=200 */</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>/* b&lt;sub&gt;1&lt;/sub&gt;=50, a.Balance=200, b&lt;sub&gt;2&lt;/sub&gt;=200 */</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td>Time</td>
<td>P2</td>
</tr>
<tr>
<td>----</td>
<td>------</td>
<td>----</td>
</tr>
<tr>
<td>Select Balance Into :b₁</td>
<td>1</td>
<td>Select Balance Into :b₂</td>
</tr>
<tr>
<td>From Account</td>
<td></td>
<td>From Account</td>
</tr>
<tr>
<td>Where Account_Id = :a</td>
<td></td>
<td>Where Account_Id = :a</td>
</tr>
<tr>
<td>/* b₁=0, a.Balance=100, b₂=0 */</td>
<td></td>
<td>/* b₂=0, a.Balance=100, b₁=0 */</td>
</tr>
<tr>
<td>b₁ = b₁-50</td>
<td>2</td>
<td>b₂ = b₂ +100</td>
</tr>
<tr>
<td>/* b₁=100, a.Balance=100, b₂=100 */</td>
<td>/* b₁=50, a.Balance=100, b₂=100 */</td>
<td></td>
</tr>
<tr>
<td>/* b₁=50, a.Balance=100, b₂=200 */</td>
<td>/* b₁=50, a.Balance=50, b₂=200 */</td>
<td></td>
</tr>
<tr>
<td>Update Account</td>
<td>5</td>
<td>Update Account</td>
</tr>
<tr>
<td>Set Balance = :b₁</td>
<td></td>
<td>Set Balance = :b₂</td>
</tr>
<tr>
<td>Where Account_Id = :a</td>
<td>6</td>
<td>Where Account_Id = :a</td>
</tr>
<tr>
<td>/* b₁=50, a.Balance=200, b₂=200 */</td>
<td>/* b₁=50, a.Balance=200, b₂=200 */</td>
<td></td>
</tr>
</tbody>
</table>

**Observation:** concurrency or parallelism may cause inconsistencies, requires concurrency control for “isolation”
void main ( ) {
    /* read user input */
    scanf ("%d %d %d", &sourceid, &targetid, &amount);
    /* subtract amount from source account */
    EXEC SQL Update Account
        Set Balance = Balance - :amount Where Account_Id = :sourceid;
    /* add amount to target account */
    EXEC SQL Update Account
        Set Balance = Balance + :amount Where Account_Id = :targetid;
    EXEC SQL Commit Work;
}
OLTP Example 1.2: Funds Transfer

```c
void main ( ) {
    /* read user input */
    scanf("%d %d %d", &sourceid, &targetid, &amount);
    /* subtract amount from source account */
    EXEC SQL Update Account
    Set Balance = Balance - :amount Where Account_Id = :sourceid;
    /* add amount to target account */
    EXEC SQL Update Account
    Set Balance = Balance + :amount Where Account_Id = :targetid;
    EXEC SQL Commit Work; } 
```

Observation: failures may cause inconsistencies, require recovery for “atomicity” and “durability”
E-Commerce Example

Shopping at Internet book store:
• client connects to the book store's server and starts browsing and querying the store's catalog
• client fills electronic shopping cart
• upon check-out client makes decision on items to purchase
• client provides information for definitive order (including credit card or cyber cash info)
• merchant's server forwards payment info to customer's bank credit or card company or cyber cash clearinghouse
• when payment is accepted, shipping of ordered items is initiated by the merchant's server and client is notified
E-Commerce Example

Shopping at Internet book store:
• client connects to the book store's server and starts browsing and querying the store's catalog
• client fills electronic shopping cart
• upon check-out client makes decision on items to purchase
• client provides information for definitive order (including credit card or cyber cash info)
• merchant's server forwards payment info to customer's bank credit or card company or cyber cash clearinghouse
• when payment is accepted, shipping of ordered items is initiated by the merchant's server and client is notified

Observations: distributed, heterogeneous system with general information/document/mail servers and transactional effects on persistent data and messages
Workflow Example

Workflows are (the computerized part of) business processes, consisting of a set of (automated or intellectual) activities with specified control and data flow between them (e.g., specified as a state chart or Petri net)

Conference travel planning:
• Select a conference, based on subject, program, time, and place. If no suitable conference is found, then the process is terminated.
• Check out the cost of the trip to this conference.
• Check out the registration fee for the conference.
• Compare total cost of attending the conference to allowed budget, and decide to attend only if the cost is within the budget.
Workflow Example

Workflows are (the computerized part of) business processes, consisting of a set of (automated or intellectual) activities with specified control and data flow between them (e.g., specified as a state chart or Petri net)

Conference travel planning:
• Select a conference, based on subject, program, time, and place. If no suitable conference is found, then the process is terminated.
• Check out the cost of the trip to this conference.
• Check out the registration fee for the conference.
• Compare total cost of attending the conference to allowed budget, and decide to attend only if the cost is within the budget.

Observations: activities spawn transactions on information servers, workflow state must be failure-resilient, long-lived workflows are not isolated
Select Conference

/ Budget:=1000;
Trials:=1;

CheckConfFee

CheckTravelCost

Check

Cost

Go

[ Cost ≤ Budget ]

Check Cost

[ Cost > Budget & Trials ≥ 3 ]

No

[ Cost > Budget & Trials < 3 ] / Trials++

Example: Travel Planning Workflow
Example: Travel Planning Workflow

![Diagram of the travel planning workflow](image-url)

1. **Select Conference**
   - \(\text{Budget}:=1000;\) \(\text{Trials}:=1;\)

2. **CheckConfFee**
   - If \(\text{ConfFound}\), \(\text{Cost}:=0\)
   - If \(!\text{ConfFound}\)

3. **CheckTravelCost**
   - If \(\text{Cost} \leq \text{Budget}\)
   - If \(\text{Cost} > \text{Budget} \& \text{Trials} \geq 3\)
   - If \(\text{Cost} > \text{Budget} \& \text{Trials} < 3\) / \(\text{Trials}++\)

4. **Check**
   - Go
   - No

5. **Check Airfare**

6. **Check Hotel**

7. **Compute Fee**

---

Example: Travel Planning Workflow

- / Budget:=1000; Trials:=1;
- / Cost =
- / Cost = ConfFee + TravelCost
- / Cost ≤ Budget
- / Cost > Budget & Trials ≥ 3
- / Cost > Budget & Trials < 3 / Trials++
Example: Travel Planning Workflow

Select Conference

/ Budget:=1000; Trials:=1;

CheckConfFee

Select Tutorials
Compute Fee

Check ConfFee

Check Airfare

Check TravelCost

Check Cost

Go

[Cost ≤ Budget]

Check Cost

[Cost > Budget & Trials ≥ 3]

No

[Cost > Budget & Trials < 3] / Trials++
Example: Travel Planning Workflow

Select Conference

/ Budget:=1000; Trials:=1;

CheckConfFee

Select Tutorials Compute Fee

Check Airfare

Check Hotel

CheckTravelCost

/ Cost = ConfFee + TravelCost

Go

[Cost ≤ Budget]

Check Cost

[Cost > Budget & Trials ≥ 3]

No

[Cost > Budget & Trials < 3] / Trials++
Example: Travel Planning Workflow

Select Conference

/ Budget:=1000;
Trials:=1;

[ConfFound] / Cost:=0

[!ConfFound] / Cost := ConfFee + TravelCost

Check Conf Fee

Select Tutorials

Compute Fee

Check Airfare

Check Hotel

Check Travel Cost

Go

[Cost ≤ Budget]

[Cost > Budget & Trials ≥ 3]

No

[Cost > Budget & Trials < 3] / Trials++
Example: Travel Planning Workflow

- **Select Conference**
  - / Budget:=1000; Trials:=1;
  - [ConfFound] / Cost:=0
  - ![ConfFound]

- **CheckConfFee**
  - Select Tutorials
  - Compute Fee

- **CheckTravelCost**
  - Check Airfare
  - Check Hotel

- **Check Cost**
  - / Cost = ConfFee + TravelCost
  - [Cost > Budget & Trials ≥ 3] / [Cost > Budget & Trials < 3] / Trials++

- **Go**
  - [Cost ≤ Budget]

- **No**
  - Check Airfare
Example: Travel Planning Workflow

Select Conference

/ Budget:=1000;
Trials:=1;

CheckConfFee

Select Tutorials
Compute Fee

[ConfFound]
/ Cost:=0

[!ConfFound]

CheckTravelCost

Check Airfare
Check Hotel

Go

[Cost ≤ Budget]

Check Cost

[Cost > Budget & Trials ≥ 3]

No

[Cost > Budget & Trials < 3] / Trials++
Example: Travel Planning Workflow

/ Budget:=1000; Trials:=1;

Select Conference

[ConfFound]
/ Cost:=0

[!ConfFound]

Check Conf Fee

Select Tutorials
Compute Fee

Check Airfare
Check Hotel

Check Travel Cost

Go

[Cost ≤ Budget]

Check Cost

[Cost > Budget & Trials ≥ 3]

No

[Cost > Budget & Trials < 3] / Trials++
Introduction

• Application Examples
• System Paradigms
  • Virtues of Transactions
  • Architecture of Database Servers
  • Lessons Learned

“If I had had more time, I could written you a shorter letter”
(Blaise Pascal)
3-Tier System Architectures

• **Clients:**
  presentation (GUI, Internet browser)

• **Application server:**
  • application programs (business objects, servlets)
  • request brokering (TP monitor, ORB, Web server)
    based on *middleware* (CORBA, DCOM, EJB, SOAP, etc.)

• **Data server:**
  database / (ADT) object / document / mail / etc. servers

Specialization to 2-Tier Client-Server Architecture:

• Client-server with “fat” clients (app on client + ODBC)
• Client-server with “thin” clients (app on server, e.g., stored proc)
3-Tier Reference Architecture

Users
Clients
Application Server

Data Server

Request
Reply

Application Program 1
Application Program 2

Request
Reply

Objects

Stored Data (Pages)

encapsulated data

exposed data
System Federations

Users

Clients

Application Servers

Data Servers
Introduction

• Application Examples
• System Paradigms

• Virtues of Transactions
• Architecture of Database Servers
• Lessons Learned

“If I had had more time, I could written you a shorter letter”
(Blaise Pascal)
ACID Properties of Transactions

• **Atomicity:**
  all-or-nothing effect,
  simple (but not completely transparent) failure handling

• **Consistency-preservation:**
  transaction abort upon consistency violation

• **Isolation:**
  only consistent data visible as if single-user mode,
  concurrency is masked to app developers

• **Durability (persistence):**
  committed effects are failure-resilient

**Transaction programming interface (“ACID contract”)**

• begin transaction
• commit transaction (“commit work” in SQL)
• rollback transaction (“rollback work” in SQL)
Requirements on Transactional Servers

Server components:

- **Concurrency Control**
  guarantees isolation
- **Recovery**:
  guarantees atomicity and durability

- **Performance**:
  high throughput (committed transactions per second)
  short response time
- **Reliability**:
  (almost) never lose data despite failures
- **Availability**:
  very short downtime
  almost continuous, 24x7, service
Introduction

- Application Examples
- System Paradigms
- Virtues of Transactions
- Architecture of Database Servers
- Lessons Learned

“If I had had more time, I could written you a shorter letter”
(Blaise Pascal)
Database System Layers

Clients

Database Server

Database

Requests

Language & Interface Layer
Query Decomposition & Optimization Layer
Query Execution Layer
Access Layer
Storage Layer

Data Accesses

Database
### Storage Structures

#### Database Page

<table>
<thead>
<tr>
<th>Page Header</th>
<th>Ben</th>
<th>55</th>
<th>Las Vegas</th>
<th>⋮</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sue</td>
<td>23</td>
<td>Seattle</td>
<td>⋮</td>
<td></td>
</tr>
<tr>
<td>Joe</td>
<td>29</td>
<td>San Antonio</td>
<td>⋮</td>
<td></td>
</tr>
</tbody>
</table>

- **Free space**
- **Slot Array**

---

#### Extent Table

- **Database**
- **Extents**

- **FORWARDING RID**
Access Structures

Search tree interface:
• lookup <index> where <indexed field> = <search key>
• lookup <index> where <indexed field> between <lower bound> and <higher bound>
Query Execution Plans

Select Name, City, Zipcode, Street
From Person
Where Age < 30
And City = "Austin"
Introduction

- Application Examples
- System Paradigms
- Virtues of Transactions
- Architecture of Database Servers
- Lessons Learned

“If I had had more time, I could written you a shorter letter”
(Blaise Pascal)
Lessons Learned

• **Benefits of ACID contract:**
  - For users: federation-wide data consistency
  - For application developers: ease of programming

• **Server obligations:**
  - Concurrency control
  - Recovery