

# Chapter 14 Database Transactions

Charles Balkon

San Jose State University

Spring 2008

# What is a Transaction?

- AKA [atomic transaction].

# What is a Transaction?

- AKA [atomic transaction].
- *logical unit of work: Do or Do Not.*

# What is a Transaction?

- AKA [atomic transaction].
- logical unit of work: *Do* or *Do Not*.
- App designer(you) creates an indivisible unit of db actions.

# What is a Transaction?

- AKA [atomic transaction].
- logical unit of work: *Do or Do Not*.
- App designer(you) creates an indivisible unit of db actions.
- **A transaction can have Accesses and Modifications.**

# The Transaction Process

- 1. Open a connection to a database
- 2. Start a transaction.
- 3. Do a list of SELECTs and or UPDATEs.
- 4. Close the transaction by COMMIT; or ROLLBACK;
- 5. Close the database connection.

# The Transaction Process

- 1. Open a connection to a database
- 2. Start a transaction.
- 3. Do a list of SELECTs and or UPDATEs.
- 4. Close the transaction by COMMIT; or ROLLBACK;
- 5. Close the database connection.

# The Transaction Process

- 1. Open a connection to a database
- 2. Start a transaction.
- 3. Do a list of **SELECTs** and or **UPDATEs**.
- 4. Close the transaction by **COMMIT**; or **ROLLBACK**;
- 5. Close the database connection.

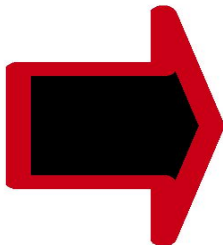
# The Transaction Process

- 1. Open a connection to a database
- 2. Start a transaction.
- 3. Do a list of SELECTs and or UPDATEs.
- 4. Close the transaction by COMMIT; or ROLLBACK;
- 5. Close the database connection.

# The Transaction Process

- 1. Open a connection to a database
- 2. Start a transaction.
- 3. Do a list of SELECTs and or UPDATEs.
- 4. Close the transaction by COMMIT; or ROLLBACK;
- 5. Close the database connection.

# Commit



- Commit = yes, save change.
- One way street
- Like when the pig becomes bacon.

# Rollback

- Rollback = Ctrl+Z, Undo.
- Your personal time machine. ( Visit DB in past )
- If you get an error during your transaction: Rollback.
- MySQL: unfinished updates or corrupting activities are not committed (db auto rollback).

## Example of a Transaction

- START TRANSACTION;
- INSERT INTO users (name, pass) VALUES ('alan', PASSWORD('ferret')) ;
- yyy
- COMMIT;

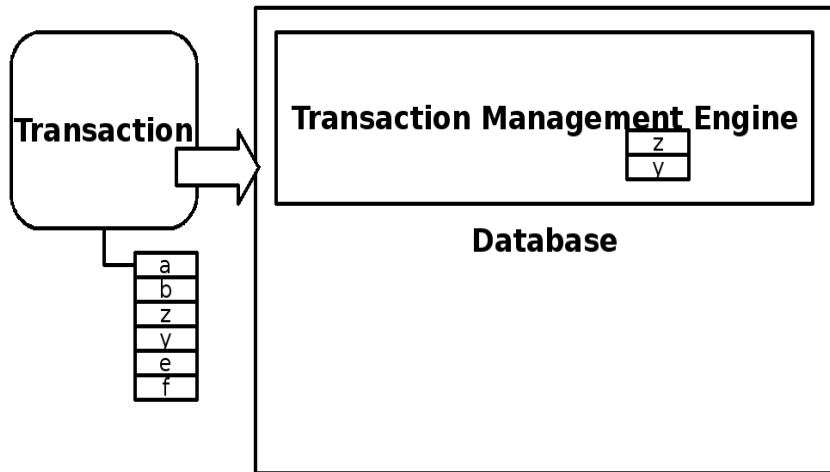
Another Example:

- BEGIN WORK;
- INSERT INTO users (name, pass) VALUES ('alan', PASSWORD('ferret')) ;
- yyy
- ROLLBACK;

### MySQL Transaction Support

*Use the InnoDB or BDB engines.*

# Idea behind DB with Transaction Management



# ACID

- A = Atomic
- C = Consistent
- I = Isolated
- D = Durable

Transaction fails ACID == BAD

# Atomicity

- one complete unit
- smallest
- no reduce
- property: updates by transaction are all used or not

- db should be consistent previous T
- db should be consistent after T
- property: after applying ACID T db is consistent.

- a  $T_a$  not affected by uncommitted  $T_b, T_c, T_d$
- property: A transaction is not affected by uncommitted transactions.

- Transaction fails ACID == BAD
- property: committed transaction means all updates are permanent.

- Types: read,write
- sizes: database, table, row, column, field, index, system resource.

- Allow other T to read.
- Nonexclusive.
- RL can upgrade to WL if no other T has RL.

# Write Lock

- Only me can use.
- To get it - no RLs or WLs on it before me.
- WL can be downgraded to RL at any time.

# Problems with Nonisolated Transactions

- concurrent executions = major problem.
- All at the same exact time or interleaved:
- Ta updates account balance to 50.00
- Tb updates account balance to 30.00
- Tc updates account balance to 5.00

# Last Update Problem

- multiple transactions calculate different balances
- the last T to update bank account balance wins

# Dirty Read Problem

- Ta reads balance 20.00
- Ta updates to 25.00
- Tb reads balance and uses 25.00 (DIRTY READ)
- Ta aborts and rollback
- Tb adds 10.00 to account
- Tb updates with 35.00
- ACTUALLY: account balance should be 30.00

# Incorrect Summary Problem

- Ta read balA (10.00), add to balA(5.00)
- Ta update balA (15.00)
- Tb get sum of balA(15.00) and balB(50.00) (sum BAD, 65.00)
- Ta read balB(50.00), sub from balB(5.00)
- Ta update balB(45.00)
- ACTUALLY: account sum of balA+balB = 60.00

# Unrepeatable Read Problem

- Ta read balA (10.00)
- Tb update balA (0.00)
- Ta read balA (0.00)
- PROBLEM: Ta sees 2 different values of balA during the same run

# Phantom Problem

- Ta get all total: balA(10.00), balB(100.00), balC(1000.00)
- Tb add a new balD(1.11)
- Ta get all total: balA,balB,balC,balD = (1111.11)
- Tb rollback - balD deleted, balD is a ghost
- PROBLEM: Ta total is different values depending when it is run
- Wrong balance, Real balance = (1110.00)

- concurrent executions = major problem.
- SiteVisitor1....SiteVisitorX
- watch, break of deadlocks possible: co\$t.

## 2 phase locking protocol

- aka: 2PL.
- 1st phase: growing.
- 2nd phase: shrinking.
- NO Phase Mixing
- Easy to enforce.

## 2PL - Growing Phase

- Growing phase:
- T get more L
- T SharedL++ = ExclusiveL

## 2PL - Shrinking Phase

- Shrinking phase:
- T locks-
- ExclusiveL- = SharedL

# More 2 phase locking protocol

- basic two-phase locking ( basic 2PL ):
- T requests L or upgradeL in Shrinking = abort!
  
- conservative two-phase locking ( conservative 2PL ):
- T req - all L at start of T
- Problem: L all objects MIGHT use, Concurrency suffers

# strict 2 phase locking protocol

- most popular
- hold locks until end of T
- release locks on Commit
- release locks on Rollback

- Order of the actions of a Transaction
- Conflict - action of  $T_a$  and  $T_b$  use same obj and an update occurs.
- Schedule is SERIAL = no interleaving of actions of multiple  $T_s$

# Serializable Schedule

- The goal of T Management exec all T as SS.
- There must be a way to reorder actions of a T =Original.
- want serializable schedule = isolated runs

# Not using transactions

- can be 3-5 times faster.

# Not using transactions

- can be 3-5 times faster.
- **Make backups!**

# Not using transactions

- can be 3-5 times faster.
- Make backups!
- *Make backups.*

# Not using transactions

- can be 3-5 times faster.
- Make backups!
- *Make backups.*

- **MAKE BACKUPS.** ;(

# Not using transactions

- can be 3-5 times faster.
- Make backups!
- *Make backups.*

- **MAKE BACKUPS.** ;(

- Turn on binary logging.

# Not using transactions

- can be 3-5 times faster.
- Make backups!
- *Make backups.*

- **MAKE BACKUPS.** ;(

- Turn on binary logging.
- **With MySQL you can probably get 50-500 transactions per minute.**

- <http://dev.mysql.com/doc/refman/5.0/en/ansi-diff-transactions.html>.
- Principles of Database Systems with Internet and Java Applications by Greg Riccardi.
- <http://www.news.com/IBM-tops-server-speed-test/2100-10103-5458910.html>