Transactions

• Major component of database systems
• Critical for most applications; arguably more so than SQL
• Turing awards to database researchers:
  – Charles Bachman 1973
  – Edgar Codd 1981 for inventing relational dbs
  – Jim Gray 1998 for inventing transactions

Concurrency control:
Three Famous anomalies

• Dirty read
  – T reads data written by T’ while T’ has not committed
  – What can go wrong: T’ writes more data (which T has already read), or T’ aborts

• Lost update
  – Two tasks T and T’ both modify the same data
  – T and T’ both commit
  – Final state shows effects of only T, but not of T’

• Inconsistent read
  – One task T sees some but not all changes made by T’

Why Do We Need Transactions

• Concurrency control

• Recovery

In the following examples, think of a transaction as meaning a procedure.
A transaction commits when it ends successfully.
A transaction rolls back when it aborts.
Dirty Reads

Client 1:
/* transfer $100 from account 1 to account 2 */
If Account1.balance > 100
   then Account1.balance = Account1.balance - 100
      Account2.balance = Account2.balance + 100
      COMMIT
else ROLLBACK

Client 2:
/* Compute total amount */
X = Account1.balance;
Y = Account2.balance;
Z = X + Y;
Print(Z);
COMMIT

What goes wrong?

Lost Updates

Client 1:
UPDATE Product
SET Price = Price – 1.99
WHERE pname = 'Gizmo'

Client 2:
UPDATE Product
SET Price = Price*0.5
WHERE pname='Gizmo'

Two different users attempt to apply a discount. Will it work?

Inconsistent Read

Client 1:
UPDATE Products
SET quantity = quantity + 5
WHERE product = 'gizmo'

Client 2:
SELECT sum(quantity)
FROM Product

Client 2:
UPDATE Products
SET quantity = quantity - 5
WHERE product = 'gadget'

Note: this is a form of dirty read
Protection against crashes

Client 1:

UPDATE Products
SET quantity = quantity + 5
WHERE product = 'gizmo'

UPDATE Products
SET quantity = quantity - 5
WHERE product = 'gadget'

What’s wrong?

Definition

• A transaction = one or more operations, which reflects a single real-world transition
  – In the real world, this happened completely or not at all

• Examples
  – Transfer money between accounts
  – Purchase a group of products
  – Register for a class (either waitlist or allocated)

• If grouped in transactions, all problems in previous slides disappear

Transactions in SQL

• In “ad-hoc” SQL:
  – Default: each statement = one transaction

• In a program:
  START TRANSACTION
  [SQL statements]
  COMMIT or ROLLBACK (=ABORT)

Revised Code

Client 1:

START TRANSACTION
UPDATE Product
SET Price = Price – 1.99
WHERE pname = ‘Gizmo’
COMMIT

Client 2:

START TRANSACTION
UPDATE Product
SET Price = Price*0.5
WHERE pname = ‘Gizmo’
COMMIT

Now it works like a charm
Transaction Properties

ACID

- **Atomic**
  - State shows either all the effects of transaction, or none of them
- **Consistent**
  - Transaction moves from a state where integrity holds, to another where integrity holds
- **Isolated**
  - Effect of transactions is the same as transactions running one after another (i.e., looks like batch mode)
- **Durable**
  - Once a transaction has committed, its effects remain in the database

ACID: Atomicity

- Two possible outcomes for a transaction
  - It *commits*: all the changes are made
  - It *aborts*: no changes are made
- That is, transaction’s activities are all or nothing

ACID: Consistency

- The state of the tables is restricted by integrity constraints
  - Account number is unique
  - Stock amount can’t be negative
  - Sum of debits and of credits is 0
- Constraints may be *explicit* or *implicit*
- How consistency is achieved:
  - Programmer makes sure a transaction takes a consistent state to a consistent state
  - The system makes sure that the transaction is atomic

ACID: Isolation

- A transaction executes concurrently with other transaction
- Isolation: the effect is as if each transaction executes in isolation of the others
ACID: Durability

- The effect of a transaction must continue to exist after the transaction, or the whole program has terminated
- Means: write data to disk (stable storage)

ROLLBACK

- If the app gets to a place where it can’t complete the transaction successfully, it can execute ROLLBACK
- This causes the system to “abort” the transaction
  - The database returns to the state without any of the previous changes made by activity of the transaction

Reasons for Rollback

- User changes their mind (“ctl-C”/cancel)
- Explicit in program, when application program finds a problem
  - e.g. when qty on hand < qty being sold
- System-initiated abort
  - System crash
  - Housekeeping
    - e.g. due to timeouts

READ-ONLY Transactions

```
Client 1: START TRANSACTION
        INSERT INTO SmallProduct(name, price)
        SELECT p.name, p.price
        FROM Product p
        WHERE p.price <= 0.99
        DELETE Product p
        WHERE p.price <= 0.99
        COMMIT

Client 2: SET TRANSACTION READ ONLY
        START TRANSACTION
        SELECT count(*)
        FROM Product
        SELECT count(*)
        FROM SmallProduct
        COMMIT
```

Makes it faster
Isolation Levels in SQL

1. “Dirty reads”
   SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
2. “Committed reads”
   SET TRANSACTION ISOLATION LEVEL READ COMMITTED
3. “Repeatable reads”
   SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
4. Serializable transactions (default):
   SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

Isolation Level: Dirty Reads

function AllocateSeat( %request)
SET ISOLATION LEVEL READ UNCOMMITTED
START TRANSACTION
Let x = SELECT Seat.occupied
FROM Seat
WHERE Seat.number = %request
If (x = 1) /* occupied */ ROLLBACK
UPDATE Seat
SET occupied = 1
WHERE Seat.number = %request
COMMIT

What can go wrong?

Isolation Level: Read Committed

Are dirty reads OK here?

What if we switch the two updates?

function TransferMoney( %amount, %acc1, %acc2)
START TRANSACTION
Let x = SELECT Account.balance
FROM Account
WHERE Account.number = %acc1
If (x < %amount) ROLLBACK
UPDATE Account
SET balance = balance+%amount
WHERE Account.number = %acc2
UPDATE Account
SET balance = balance-%amount
WHERE Account.number = %acc1
COMMIT

Stronger than READ UNCOMMITTED

It is possible to read twice, and get different values

/* . . . . . More stuff here . . . . */

Let y = SELECT Seat.occupied
FROM Seat
WHERE Seat.number = %request
/* we may have x ≠ y */

Isolation Level: Repeatable Read

Stronger than
READ COMMITTED

May see incompatible
values:

another txn transfers
from acc. 55555 to
77777

Let \( x = \) SELECT Account.amount
FROM Account
WHERE Account.number = '555555'
/* . . . . . More stuff here . . . . */
Let \( y = \) SELECT Account.amount
FROM Account
WHERE Account.number = '777777'
/* we may have a wrong x+y ! */

Isolation Level: Serializable

Default

WILL STUDY IN DETAILS IN A WEEK

The Mechanics of Disk

Mechanical characteristics:
- Rotation speed (5400RPM)
- Number of platters (1-30)
- Number of tracks (\(<=10000\))
- Number of bytes/track (10^5)

Unit of read or write:
disk block
Once in memory:
page
Typically: 4k or 8k or 16k

Disk Access Characteristics

- Disk latency = time between when command is issued and
  when data is in memory
  - Disk latency = seek time + rotational latency
    - Seek time = time for the head to reach cylinder
      - 10ms - 40ms
    - Rotational latency = time for the sector to rotate
      - Rotation time ~ 10ms
      - Average latency ~ 10ms/2
- Transfer time = typically 40MB/s
- Disks read/write one block at a time
RAID

Several disks that work in parallel
- Redundancy: use parity to recover from disk failure
- Speed: read from several disks at once

Various configurations (called levels):
- RAID 1 = mirror
- RAID 4 = n disks + 1 parity disk
- RAID 5 = n+1 disks, assign parity blocks round robin
- RAID 6 = “Hamming codes”

Buffer Manager

Needs to decide on page replacement policy
- LRU
- Clock algorithm

Both work well in OS, but not always in DB

Enables the higher levels of the DBMS to assume that the needed data is in main memory.

Buffer Management in a DBMS

Page Requests from Higher Levels

Buffer Manager Needs to decide on page replacement policy
- LRU
- Clock algorithm

Both work well in OS, but not always in DB

Enables the higher levels of the DBMS to assume that the needed data is in main memory.

Buffer Management in a DBMS

Page Requests from Higher Levels

Buffer Manager Needs to decide on page replacement policy
- LRU
- Clock algorithm

Both work well in OS, but not always in DB

Enables the higher levels of the DBMS to assume that the needed data is in main memory.

Buffer Management in a DBMS

Page Requests from Higher Levels

Buffer Manager Needs to decide on page replacement policy
- LRU
- Clock algorithm

Both work well in OS, but not always in DB

Enables the higher levels of the DBMS to assume that the needed data is in main memory.

Buffer Management in a DBMS

Page Requests from Higher Levels

Buffer Manager Needs to decide on page replacement policy
- LRU
- Clock algorithm

Both work well in OS, but not always in DB

Enables the higher levels of the DBMS to assume that the needed data is in main memory.

Buffer Management in a DBMS

Page Requests from Higher Levels

Buffer Manager Needs to decide on page replacement policy
- LRU
- Clock algorithm

Both work well in OS, but not always in DB

Enables the higher levels of the DBMS to assume that the needed data is in main memory.

Buffer Management in a DBMS

Page Requests from Higher Levels

Buffer Manager Needs to decide on page replacement policy
- LRU
- Clock algorithm

Both work well in OS, but not always in DB

Enables the higher levels of the DBMS to assume that the needed data is in main memory.

Buffer Management in a DBMS

Page Requests from Higher Levels

Buffer Manager Needs to decide on page replacement policy
- LRU
- Clock algorithm

Both work well in OS, but not always in DB

Enables the higher levels of the DBMS to assume that the needed data is in main memory.

Buffer Management in a DBMS

Page Requests from Higher Levels

Buffer Manager Needs to decide on page replacement policy
- LRU
- Clock algorithm

Both work well in OS, but not always in DB

Enables the higher levels of the DBMS to assume that the needed data is in main memory.

Buffer Management in a DBMS

Page Requests from Higher Levels

Buffer Manager Needs to decide on page replacement policy
- LRU
- Clock algorithm

Both work well in OS, but not always in DB

Enables the higher levels of the DBMS to assume that the needed data is in main memory.

Buffer Management in a DBMS

Page Requests from Higher Levels

Buffer Manager Needs to decide on page replacement policy
- LRU
- Clock algorithm

Both work well in OS, but not always in DB

Enables the higher levels of the DBMS to assume that the needed data is in main memory.

Buffer Management in a DBMS

Page Requests from Higher Levels

Buffer Manager Needs to decide on page replacement policy
- LRU
- Clock algorithm

Both work well in OS, but not always in DB

Enables the higher levels of the DBMS to assume that the needed data is in main memory.

Buffer Management in a DBMS

Page Requests from Higher Levels

Buffer Manager Needs to decide on page replacement policy
- LRU
- Clock algorithm

Both work well in OS, but not always in DB

Enables the higher levels of the DBMS to assume that the needed data is in main memory.

Buffer Management in a DBMS

Page Requests from Higher Levels

Buffer Manager Needs to decide on page replacement policy
- LRU
- Clock algorithm

Both work well in OS, but not always in DB

Enables the higher levels of the DBMS to assume that the needed data is in main memory.
Buffer Manager

Why not use the Operating System for the task??

Main reason: need fine grained control for transactions

Other reasons:
- DBMS may be able to anticipate access patterns
- Hence, may also be able to perform prefetching
- DBMS needs the ability to force pages to disk, for recovery purposes

Transaction Management and the Buffer Manager

The transaction manager operates on the buffer pool

- **Recovery**: ‘log-file write-ahead’, then careful policy about which pages to force to disk
- **Concurrency control**: locks at the page level, multiversion concurrency control

Will discuss details during the next few lectures