Chapter 15

Transaction Management

Outline

- Transaction basics
- Concurrency control
- Transaction design issues
- Workflow management

Transaction Definition

- Collection of database operations processed as one logical unit of work
- Prevents lost data due to
  - Interference among concurrent users
  - Failures
- Supports daily operations of an organization

ATM Example (No Transaction)

```
CREATE PROCEDURE withdraw(IN id INT, IN amt NUMERIC(18,2))
BEGIN
  DECLARE curamt numeric(18,2)

  SELECT balance INTO curamt
  FROM account WHERE account_id = id;
  IF NOT curamt IS NULL THEN
    IF curamt - amt >= 0 THEN
      UPDATE account SET balance = curamt - amt
      WHERE account_id = id;
    END IF;
  END IF;

END;
```

What can go wrong if two people execute this procedure against the same account, simultaneously?

account

account_id	balance
100	5000.00
101	3000.00

ATM Transaction Example

```
CREATE PROCEDURE withdraw(IN id INT, IN amt NUMERIC(18,2))
BEGIN
  DECLARE curamt numeric(18,2)
  START TRANSACTION;
  SELECT balance INTO curamt
  FROM account WHERE account_id = id;
  IF NOT curamt IS NULL THEN
    IF curamt - amt >= 0 THEN
      UPDATE account SET balance = curamt - amt
      WHERE account_id = id;
    END IF;
  END IF;
  COMMIT;
END;
```

Transaction Properties

- Atomic
  - All updates in a given transaction either succeed or fail as a unit
- Consistent
  - Because of the atomic and isolated properties, a properly designed transaction is guaranteed to take the database from one valid state to another

Transaction Properties

- Isolated
  - When transactions run concurrently, one transaction cannot "see" updates made by another concurrent transaction
- Durable
  - A transaction's updates persist upon completion, even if certain types of system failure occur

Concurrency Control

- Problem definition
- Concurrency control problems
- Concurrency control tools

Concurrency Control Problem

- Objective:
  - Maximize **throughput**: number of transactions processed per unit time
- Constraint:
  - No interference between transactions
  - Effect should be the same as if transactions were executed sequentially

Lost Update Problem

Transaction A	Time	Transaction B
Read SR (10)	T_1	
	T_2	Read SR (10)
SR = SR -1	T_3	
	T_4	SR = SR -1
Write SR (9)	T_5	
	T_6	Write SR (9)

Dirty Read Problem

Transaction A	Time	Transaction B
Read SR (10)	T_1	
SR = SR - 1	T_2	
Write SR (9)	T_3	
	T_4	Read SR (9)
ROLLBACK	T_5	

Phantom Reads

- Interference causes inconsistency among multiple retrievals of a subset of data
- Examples:
  - Incorrect summary

Preventing Concurrency Problems

- Two approaches:
  - Pessimistic (using locking)
  - Optimistic (using versioning)

Locking Fundamentals

- Obtain lock before accessing an item
- Wait if a conflicting lock is held
  - Shared lock: blocks exclusive locks
  - Exclusive lock: blocks shared and exclusive locks

Two Phase Locking

Within a transaction:

- Database acquires
  - shared locks for SELECT statements
  - exclusive locks for INSERT/UPDATE/DELETE statements
- Locks are held until COMMIT / ROLLBACK

Deadlock (Mutual Waiting)

Transaction A	**Time**	**Transaction B**
XLock SR_1	T_1	
	T_2	XLock SR_2
XLock SR_2 (wait)	T_3	
	T_4	XLock SR_1 (wait)

Isolation Levels

- Degree to which a transaction is separated from the actions of other transactions
- Balance concurrency control overhead with interference problems
- Some transactions can tolerate uncommitted dependency and inconsistent retrieval problems
- Use the SET TRANSACTION ISOLATION LEVEL statement

SQL Isolation Levels

Level	XLocks	SLocks	Interference
Read uncommitted	Long	None	All
Read committed	Long	Short	All except dirty read
Repeatable read	Long	Long	Phantom reads
Serializable	Long	Long	None

Improving Transaction Performance

- Adjust transaction boundaries
- Reorder transaction operations

Adjust Transaction Boundaries

- START TRANSACTION;
  SELECT * FROM ...
  ... prompt user for info ...
  UPDATE ...
  COMMIT TRANSACTION
- Including user interaction in transaction boundaries is never a good idea

Adjust Transaction Boundaries

- Break original transaction into two:

```
START TRANSACTION;
SELECT * FROM ... ;
COMMIT;
... prompt user for info ...
START TRANSACTION;
SELECT * FROM ... ; -- still ok to proceed?
UPDATE ... ;
COMMIT TRANSACTION;
```

Reorder Transaction Operations

Attempt #1

```
START TRANSACTION;
SELECT balance
   FROM account
   WHERE id = ?;
IF balance – amt_withdraw < 0
    ROLLBACK
ELSE
   UPDATE account
   SET balance = balance – amt_withdraw
   WHERE id = ?;
   COMMIT
END IF
```

Attempt #2

```
START TRANSACTION;
UPDATE account
SET balance = balance – amt_withdraw
WHERE id = ?;
SELECT balance
FROM account
WHERE id = ?;
IF balance < 0
   ROLLBACK
ELSE
   COMMIT
END IF
```

Reordering Operations

- What isolation levels are required for the transactions on the previous slide to work properly?

MySQL Demo Script

```
SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;
START TRANSACTION;
SELECT * FROM account WHERE id = 101;
UPDATE account SET balance = balance - 100 WHERE id = 101;
COMMIT;
```

Optimistic Approaches

- Assumes conflicts are rare
- No locks
- Often uses **row versioning**
- Check for conflicts
  - After each read and write, or
  - At end of transaction
- Evaluation
  - Less overhead, but more performance variation

MySQL Concurrency Control

- Hybrid Approach
- **Writes** place exclusive locks
- **Reads** use both versioning and shared locks
  - Read Committed and Repeatable Read: Versioning
  - Serializable: Shared locks

MySQL Multi-Row Versioning

- MySQL takes a "snapshot" of rows to provide a consistent read for transactions running in Read Committed and Repeatable Read
- Read Committed: Snapshot taken for each read
- Repeatable Read: Snapshot taken upon first read
- See MySQL Consistent Reads:
  - https://dev.mysql.com/doc/refman/8.0/en/innodb-consistent-read.html

Python and Transactions

- Connection object has commit() and rollback() methods
- Failure to use commit() means that no insert/update/delete results will be committed to database
- Can turn on autocommit to avoid using commit() and rollback()

Summary

- Transaction: user-defined collection of work
- DBMSs support ACID properties
- Knowledge of concurrency control and recovery important for managing databases
- Transaction design issues are important
- Transaction processing is an important part of workflow management