Computer Science 202

Transaction Management and Concurrency Control

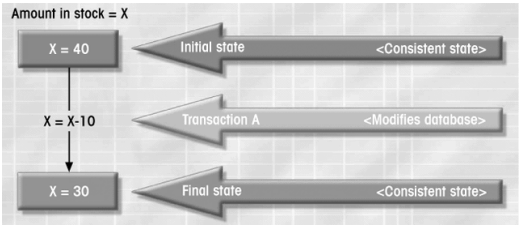
Objectives

- To learn what a transaction is.
- To learn how an incomplete transaction can yield an inconsistent database.
- To learn how an inconsistent database can be avoided through proper transaction management.
- To learn what concurrency control is.
- To learn how concurrency control failure affects a database.
- To learn how concurrency control is accomplished.
- To learn what database recovery management is.
- To understand the use of transaction logs in recovery management.

What is a Transaction?

- A logical unit of work on a database
  - Read (select) and write (Update/insert/delete)
- Database request is equivalent to a single SQL statement
- A transaction can consist of many DB Requests
- A transaction must be completed, or must be aborted
- Consistency of a database is paramount
- DB should be maintained in a consistent state

Example Transaction



So what's the problem?

- To ensure consistency
  - A Transaction must start with the DB in a known state
  - The transaction must complete and leave the DB in a known state
- A transaction is likely to modify DB contents
  - Protection must be given for the areas of the DB that the transaction works with.

Transaction Evaluation (I)

- Examine current account balance

```
SELECT ACC_NUM, ACC_BALANCE
FROM CHECKACC
WHERE ACC_NUM = '0908110638';
```

- Consistent state after transaction
- No changes made to Database

Transaction Evaluation (II)

- Register credit sale of 100 units of product X to customer Y for R500

```
UPDATE PRODUCT
SET PROD_QOH = PROD_QOH - 100
WHERE PROD_CODE = 'X';
UPDATE ACCT_RECEIVABLE
SET ACCT_BALANCE = ACCT_BALANCE + 500
WHERE ACCT_NUM = 'Y';
```

- Consistent state only if both transactions are fully completed
- DBMS doesn't guarantee that the semantic meaning of a transaction represents real-world event

Transaction Properties

- Atomicity
  - All parts of must be completed OR none
  - Indivisible Work unit
- Durability
  - On completion DB is in a consistent state
- Serializability
  - Concurrent execution of multiple transactions
  - Concurrency can mean that transactions are handled in a serial manner
- Isolation
  - Data used in a transaction cannot be used by another un till the first completes

Database Properties

- Single User
  - Automatically ensures serializability and isolation, by design (only single user)
  - Durability & Atomicity need to be provided by the DBMS
- Multi User
  - Multiple concurrent transactions
  - DBMS must enforce serializability and isolation
  - Provide concurrency control management

SQL Transaction Management

- Transaction support
  - COMMIT
  - ROLLBACK
- User initiated transaction sequence must continue until:
  - COMMIT statement is reached
  - ROLLBACK statement is reached
  - End of a program reached (COMMIT)
  - Abnormal program termination (ROLLBACK)

SQL Example

```
BEGIN TRANSACTION; --Not ALL support THIS

UPDATE PRODUCT
   SET PROD_QOH = PROD_QOH - 100
   WHERE PROD_CODE = 'X';

UPDATE ACCT_RECEIVABLE
   SET ACCT_BALANCE = ACCT_BALANCE + 500
   WHERE ACCT_NUM = 'Y';

COMMIT;
```


Transaction Log

- Transaction log keeps track of all updates affecting the database
- Log is maintained by the DBMS
- Used for DB recovery in the event of a ROLLBACK
- Can be used by some DBMS for recovery following a crash
- Used to help ensure Durability of Transactions

Transaction Log

- Transaction log holds information to allow a recovery

TRL ID	TRX NUM	PREV PTR	NEXT PTR	OPERATON	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
341	101	Null	352	START	*** Start transaction				
352	101	341	363	UPDATE	PRODUCT	345TYX	PROD_QOH	243	143
363	101	352	365	UPDATE	ACCT_RECEIVABLE	60120010	ACCT_BALANCE	1200	4700
365	101	363	Null	COMMIT	**** End transaction				

TRL ID = Transaction log record ID **PTR** = Pointer to a transaction log record ID
TRX NUM = Transaction number
(Note: The transaction number is automatically assigned by the DBMS.)

Concurrency Control

- Ensures serializability of transactions
- Protects against problems that could arise from simultaneous execution
  - Data Integrity
  - Consistency
- Handles
  - Lost Updates
  - Uncommitted Data
  - Inconsistent Data Retrievals

Concurrency Issues

- Lost Updates
  - An update is lost and the result is Erroneous data being retrieved
- Uncommitted Data
  - Two concurrent transactions
  - T1 Rolls back after T2 Has already accessed updated information
- Inconsistent Retrievals
  - Usually with Aggregation functions
  - T2 performs an aggregation while T1 is Updating values

Lost Updates

Normal Transaction

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	135
4	T2	Read PROD_QOH	135
5	T2	PROD_QOH = 135 - 30	
6	T2	Write PROD_QOH	105

Lost Updates

Lost Update results in Error

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T2	Read PROD_QOH	35
3	T1	PROD_QOH = 35 + 100	
4	T2	PROD_QOH = 35 - 30	
5	T1	Write PROD_QOH **(Lost update)**	135
6	T2	Write PROD_QOH	5

Uncommitted Data

Correct Processing

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	135
4	T1	*****ROLLBACK *****	35
5	T2	Read PROD_QOH	35
6	T2	PROD_QOH = 35 - 30	
7	T2	Write PROD_QOH	5

Uncommitted Data

Problems resulting from Uncommitted Data

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Read PROD_QOH	35
2	T1	PROD_QOH = 35 + 100	
3	T1	Write PROD_QOH	135
4	T2	Read PROD_QOH (Read uncommitted data)	135
5	T2	PROD_QOH = 135 - 30	
6	T1	***** ROLLBACK *****	35
7	T2	Write PROD_QOH	105

Inconsistent Data Retrieval

Retrieval during an Update

TRANSACTION T1	TRANSACTION T2
SELECT SUM(PROD_QOH) FROM PRODUCT	UPDATE PRODUCT SET PROD_QOH = PROD_QOH + 30 WHERE PROD_CODE = '125TYZ'
	UPDATE PRODUCT SET PROD_QOH = PROD_QOH - 30 WHERE PROD_CODE = '345TYX'
	COMMIT;

Inconsistent Data Retrieval

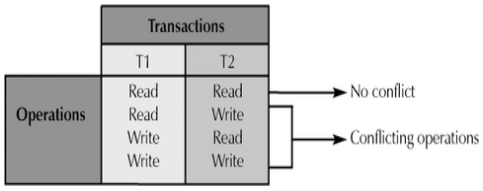
Resultant Output

	BEFORE	AFTER
PROD_CODE	PROD_QOH	PROD_QOH
104XCV	100	100
110YGH	120	120
125TYZ	70	(70 + 30) → 100
345TYX	35	(35 - 30) → 5
350TYX	100	100
355TYX	30	30
Total	**455**	**455**

The Scheduler

- Scheduler provides overall management of the concurrency process
- Ordering of transactions → serializability
- Ordering based on Control Algorithms
  - Locking
  - Time stamping
- Why Not use FIFO ordering ?
  - NOT effective use of CPU
  - Poor Response times

Conflicting Operations



Locking Methods

- A Lock guarantees exclusive access to a DB object
- Prevents access to inconsistent Data
- Automatically managed by DBMS
- A *Lock Manager* process is responsible for all Locking within the DB
- Lock Granularity can have a big effect on performance

Locking granularity

- The Level at which the locks are applied
  - Database Level
  - Table Level
  - Page Level
  - Row Level
  - Field Level
- Locks can be of two types
  - Binary
  - Shared/Exclusive

Database Level

- Entire Database is locked
- Prevents use of **any** other tables by transactions T2, T3…. until T1 has completed
- ***Very Slow*** performance

Database Level

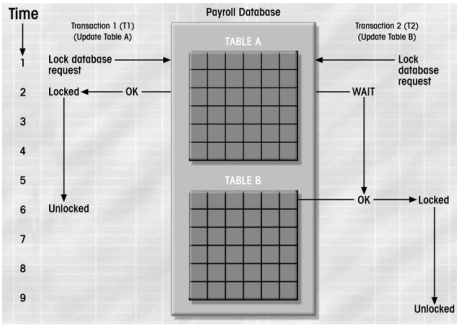
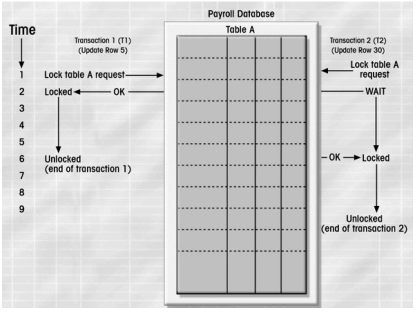


Table Level

- Locks are applied to specific tables
- T2 can access tables within the DB provided that they are not locked by T1
- Better performance than DB Level locks
- Can still be problematic with multiple accesses to a Table
- Slowdown even if T2 and T1 are accessing different portions of a table.

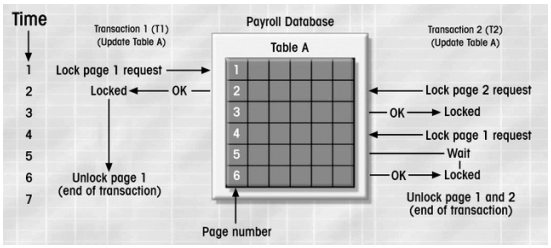
Table Level



Page Level

- Locks are specific to a physical disk page
- Disk page is equivalent to a storage block on disk
- Size of page is dependant on underlying HW, OS and File system
  - 4K , 8K , 16K , 32K
- T2 cannot access a page locked by T1
- T2 CAN access the same table if the portion needed is in a different disk page

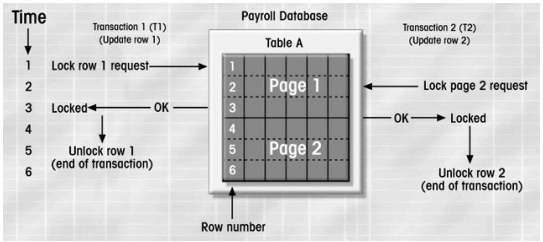
Page Level



Row Level

- Non restrictive locking mechanism
- Locks are restricted to the particular ROW that T1, may be using. T2 May access the rest of the DB.
- Higher granularity results in increased overhead

Row Level



Field Level

- Allows Transactions to access the same row
- Provides locking on fields within the row
- Extremely high overhead

Binary Locks

- Only has two states
  - Locked
  - Unlocked
- Applies to all levels of granularity
- DBMS manages the locking process
- Binary locks are generally exclusive

Binary Locks

TIME	TRANSACTION	STEP	STORED VALUE
1	T1	Lock PRODUCT	
2	T1	Read PROD_QOH	35
3	T1	PROD_QOH = 35 + 100	
4	T1	Write PROD_QOH	135
5	T1	Unlock PRODUCT	
6	T2	Lock PRODUCT	
7	T2	Read PROD_QOH	135
8	T2	PROD_QOH = 135 - 30	
9	T2	Write PROD_QOH	105
10	T2	Unlock PRODUCT	

Shared/Exclusive Locks

- Exclusive Locks
  - Only the locking transaction has access
  - Used for write operations
- Shared Locks
  - Concurrent transactions may have READ access to a locked object
  - No conflict provided all transactions are READ
  - Cannot be issued if an exclusive lock for the object exists

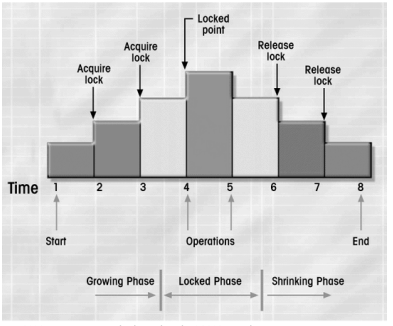
Shared/Exclusive Locks

- Locking states
  - READ_LOCK
  - WRITE_LOCK
  - UNLOCK
- Locking problems
  - Transactions may not be serializable
    - Can be solved with TWO-PHASE locking
  - Scheduler may create deadlocks

Two-Phase Locking

- Growing phase
- Shrinking phase
- Governing rules
  - Two transactions cannot have conflicting locks
  - No unlock operation can precede a lock operation in the same transaction
  - No data objects are updated until all locks are obtained

Two-Phase Locking



Deadlocks

- Occurs when two transactions are waiting for the other to unlock data
  - T1 = Accesses DATA X & DATA Y
  - T2 = Accesses DATA Y & DATA X
- Also known as a '*deadly embrace*'

Deadlock Example

TIME	TRANSACTION	REPLY	LOCK STATUS	
0			Data X	Data Y
			Unlocked	Unlocked
1	T1:LOCK(X)	OK	Locked	Unlocked
2	T2: LOCK(Y)	OK	Locked	Locked
3	T1:LOCK(Y)	WAIT	Locked	Locked
4	T2:LOCK(X)	WAIT	Locked	Locked
5	T1:LOCK(Y)	WAIT	Locked	Locked
6	T2:LOCK(X)	WAIT	Locked	Locked
7	T1:LOCK(Y)	WAIT	Locked	Locked
8	T2:LOCK(X)	WAIT	Locked	Locked
9	T1:LOCK(Y)	WAIT	Locked	Locked
...	.....	.....	.....	.....
...	.....	.....	.....	.....
...	.....	.....	.....	.....
...	.....	.....	.....	.....

Deadlock

Deadlock Control

- Deadlock Prevention
  - New locks are aborted if there is a likelihood they would cause a deadlock
  - Transaction re-scheduled
- Deadlock Detection
  - DBMS checks DB for deadlocks
  - In the case of a deadlock one transaction is aborted and then restarted
- Deadlock Avoidance
  - Transaction must obtain ALL locks it needs prior to execution (two-phase locking)
  - Much slower due to serialization of transactions

Time stamping

- Timestamps can be used to assist with scheduling algorithms
- Each transaction is assigned a unique stamp based on the order of submission to DBMS
- R/W operations can take place provided they have the same timestamp
- Transactions are serialized based on timestamp order
- Disadvantages
  - Increased admin
  - Increased memory (Last read & Last Update)

Optimistic Methods

- Assumption that most transactions to do not conflict
- Transaction executed without restrictions until committed
- Phases:
  - Read Phase
  - Validation Phase
  - Write Phase

Database Recovery Management

- Restores a database from an inconsistent to a previously consistent state
- Based on the atomic transaction property
- Level of backup
  - Full backup
  - Differential
  - Transaction log

Database Failures

- Software
  - Viruses, Operating System, DBMS
- Hardware
  - Hard disks, Memory, Networks
- Programming Exemption
  - Abnormal termination, premature termination
- Transaction
  - Deadlock results in termination of a transaction
- External
  - Fire, Flood, Theft

Transaction Recovery

- Write-Ahead Logging
  - Logs written BEFORE data is modified
- Redundant transaction logs
  - Multiple copies of transaction log maintained
- Buffers
  - Memory buffers lead to faster processing
- Checkpoints
  - Updated buffers committed to disk
  - Registered in Transaction log
  - Buffers and Physical copies synchronised

Transaction Recovery

- Deferred-write and Deferred-update
  - Changes are written to the transaction log
  - Database updated after transaction reaches commit point
- Write-through
  - Immediately updated by during execution
  - Before the transaction reaches its commit point
  - Transaction log also updated
  - Transaction fails, database uses log information to ROLLBACK