# TOPIC 5

# Transaction Management

**Upon completion this topic, students should be able to:**

5.1 Demonstrate database transaction management

CLO 1 : Apply fundamental of Database Management System (DBMS), relational data model and normalization concepts in database development process.

CLO 2 : Show a well-structured database using the database query to manipulate a database with an appropriate commercial Database Management System.

# What is a Transaction?

- A logical unit of work on a database
  - An entire program
  - A portion of a program
  - A single command
- The entire series of steps necessary to accomplish a logical unit of work
- Successful transactions change the database from one CONSISTENT STATE to another

  (One where all data integrity constraints are satisfied)

# Example of a Transaction

- Updating a Record
  - Locate the Record on Disk
  - Bring record into Buffer
  - Update Data in the Buffer
  - Writing Data Back to Disk

# Transaction processing systems category

a. Batch transaction processing system
   - is an efficient way of processing large volumes of data. where a group of transactions is collected over a period of time. Data is collected, entered, processed and then the batch results are produced.
   - e.g payroll system

b. On-line transaction processing system (OLTP)
   - is a type of data processing that consists of executing a number of transactions occurring concurrently.
   - e.g online banking, shopping, order entry, or sending text messages.

c. Real time transaction processing system
   - requires quick transaction and characterized by supplying immediate response.
   - In real time processing processor needs to very responsive and active all the time.
   - e.g ATM transaction, radar system

# 4 Properties of a Transaction

- Atomic – All or Nothing

  All parts of the transaction must be completed and committed or it must be aborted and rolled back

- Consistent

  Each user is responsible to ensure that their transaction (if executed by itself) would leave the database in a consistent state

# 4 Properties of a Transaction

- Isolation

  The final effects of multiple simultaneous transactions must be the same as if they were executed one right after the other

- Durability

  If a transaction has been committed, the DBMS must ensure that its effects are permanently recorded in the database (even if the system crashes)

# Transaction Management with SQL

- SQL Statements → Commit / Rollback
- When a transaction sequence is initiated it must continue through all succeeding SQL statements until:
  1. A Commit Statement is Reached
  2. A Rollback Statement is Reached
  3. The End of the Program is Reached (Commit)
  4. The Program  is Abnormally Terminated (Rollback)

# Example

```
BEGIN TRAN
    DECLARE @ErrorCode INT, @TranSuccessful INT
    SET @TranSuccessful = 1

    INSERT INTO tblCatalog (CatalogYear)
        VALUES('2002')
    SET @ErrorCode = @@ERROR; IF (@ErrorCode <> 0) SET @TranSuccessful = 0 –
    False

    INSERT INTO tblCatalog (CatalogYear)
        VALUES('2003')
    SET @ErrorCode = @@ERROR; IF (@ErrorCode <> 0) SET @TranSuccessful = 0 –
    False

    IF @TranSuccessful = 0
      BEGIN
        ROLLBACK TRAN
        RAISERROR ('Rolledback transaction: Insert Catalog Year.', 16,1)
      END
    ELSE
      BEGIN
        COMMIT TRAN
        PRINT 'Successfully inserted catalog years...'
      END
GO
```

# Transaction Log

- Keeps track of all transactions that update the database
  - Record for the beginning of the transaction
  - Type of operation (insert / update / delete)
  - Names of objects/tables affected by the transaction
  - Before and After Values for Updated Fields
  - Pointers to Previous and Next Transaction Log Entries for the same transaction
  - The Ending of the Transaction (Commit)
- Used for recovery in case of a Rollback

# Concurrency Control

- Coordination of simultaneous transaction execution in a multiprocessing database system
- Ensure transaction serializability in a multi-user database
- Lack of Concurrency Control can create data integrity and consistency problems:
  - Lost Updates
  - Uncommitted Data
  - Inconsistent Retrievals

# Lost Updates

| Time | Jack's Trans | Jill's Trans | Balance |
|---|---|---|---|
| T1 | Begin | | |
| T2 | Read Balance | Begin | 1000 |
| T3 | | Read Balance | 1000 |
| T4 | Bal = Bal – 50 (950) | | 1000 |
| T5 | Write Bal (950) | Bal = Bal + 100 (1100) | 950 |
| T6 | Commit | | 950 |
| T7 | | Write Bal (1100) | 1100 |
| T8 | | Commit | 1100 |

# Uncommitted Data

| Time | Deposit | Interest | Bal |
|---|---|---|---|
| T1 | Begin Transaction | | 1000 |
| T2 | Read Bal (1000) | | 1000 |
| T3 | Bal = Bal + 1000 (2000) | | 1000 |
| T4 | Write Bal (2000) | Begin Transaction | 2000 |
| T5 | | Read Bal (2000) | 2000 |
| T6 | | Bal = Bal*1.05 (2100) | 2000 |
| T7 | Rollback | | 1000 |
| T8 | | Write Bal (2100) | 2100 |
| T9 | | Commit | 2100 |

# Inconsistent Retrievals

| Time | SumBal | Transfer | Bal A | Bal B | Bal C | Sum |
|---|---|---|---|---|---|---|
| T1 | Begin Trans | | 5000 | 5000 | 5000 | |
| T2 | Sum = 0 | Begin Trans | 5000 | 5000 | 5000 | |
| T3 | Read BalA (5000) | | 5000 | 5000 | 5000 | |
| T4 | Sum = Sum + BalA (5000) | Read BalA (5000) | 5000 | 5000 | 5000 | |
| T5 | Read BalB (5000) | BalA = BalA -1000 (4000) | 5000 | 5000 | 5000 | |
| T6 | Sum = Sum+BalB (10000) | Write BalA (4000) | 4000 | 5000 | 5000 | |
| T7 | | Read BalC | 4000 | 5000 | 5000 | |
| T8 | | BalC =BalC + 1000 (6000) | 4000 | 5000 | 5000 | |
| T9 | | Write BalC (6000) | 4000 | 5000 | 6000 | |
| T10 | Read BalC | Commit | 4000 | 5000 | 6000 | |
| T11 | Sum=Sum + BalC (16000) | | 4000 | 5000 | 6000 | |
| T12 | Write Sum (16000) | | 4000 | 5000 | 6000 | 16000 |
| T13 | Commit | | 4000 | 5000 | 6000 | 16000 |

# Serial Execution of Transactions

- Serial Execution of transaction means that the transactions are performed one after another.
- No interaction between transactions - No Concurrency Control Problems
- Serial Execution will never leave the database in an inconsistent state → Every Serial Execution is considered correct (Even if a different order would cause different results)

# Serializability

- If 2 Transactions are only reading data items – They do not conflict → Order is unimportant
- If 2 Transactions operate (Read/Write) on Separate Data Items
  – They do not conflict → Order is unimportant
- If 1 Transaction Writes to a Data Item and Another Reads or Writes to the Same Data Item → The Order of Execution IS Important

# The Scheduler

- Special DBMS Program to establish the order of operations in which concurrent transactions are executes
- Interleaves the execution of database operations to ensure:

  Serializability

  Isolation of Transactions

# The Scheduler

- Bases its actions on Concurrency Control Algorithms (Locking / Time Stamping)
- Ensures the CPU is used efficiently (Scheduling Methods)
- Facilitates Data Isolation → Ensure that 2 transactions do not update the same data at the same time

# Concurrency Control Algorithms

- Locking

  A Transaction "locks" a database object to prevent another object from modifying the object

- Time-Stamping

  Assign a global unique time stamp to each transaction

- Optimistic

  Assumption that most database operations do not conflict

# Locking

- Lock guarantees exclusive use of data item to current transaction
- Prevents reading Inconsistent Data
- Lock Manager is responsible for assigning and policing the locks used by the transaction

# Locking Granularity

Indicates the level of lock use

- Database Level – Entire Database is Locked
- Table Level – Entire Table is Locked
- Page Level – Locks an Entire Diskpage (Most Frequently Used)
- Row Level – Locks Single Row of Table
- Field Level – Locks a Single Attribute of a Single Row (Rarely Done)

# Types of Locks: Binary

- Binary Locks – Lock with 2 States
  - Locked – No other transaction can use that object
  - Unlocked – Any transaction can lock and use object

  All Transactions require a Lock and Unlock Operation for Each Object Accessed (Handled by DBMS)

  - Eliminates Lost Updates
  - Too Restrictive to Yield Optimal Concurrency Conditions

# Types of Locks: Shared / Exclusive Locks

- Indicates the Nature of the Lock
- Shared Lock – Concurrent Transactions are granted READ access on the basis of a common lock
- Exclusive Lock – Access is reserved for the transaction that locked the object
- 3 States:  Unlocked, Shared (Read), Exclusive (Write)
- More Efficient Data Access Solution
- More Overhead for Lock Manager
  - Type of lock needed must be known
  - 3 Operations:
    - Read_Lock – Check to see the type of lock
    - Write_Lock – Issue a Lock
    - Unlock – Release a Lock
  - Allow Upgrading / Downgrading of Locks

# Problems with Locking

- Transaction Schedule May Not be Serializable
  - Can be solved with 2-Phase Locking
- May Cause Deadlocks
  - A deadlock is caused when 2 transactions wait for each other to unlock data

# Two Phase Locking

- Defines how transactions Acquire and Relinquish Locks

1. Growing Phase – The transaction acquires all locks (doesn't unlock any data)
2. Shrinking Phase – The transaction releases locks (doesn't lock any additional data)

- Transactions acquire all locks it needs until it reaches locked point
- When locked, data is modified and locks are released

# Deadlocks

- Occur when 2 transactions exist in the following mode:
  T1 = access data item X and Y
  T2 = Access data items Y and X

If T1 does not unlock Y, T2 cannot begin
If T2 does not unlock X, T1 cannot continue

T1 & T2 wait indefinitely for each other to unlock data

- Deadlocks are only possible if a transactions wants an Exclusive Lock (No Deadlocks on Shared Locks)

# Controlling Deadlocks

- Prevention – A transaction requesting a new lock is aborted if there is the possibility of a deadlock – Transaction is rolled back, Locks are released, Transaction is rescheduled
- Detection – Periodically test the database for deadlocks. If a deadlock is found, abort / rollback one of the transactions
- Avoidance – Requires a transaction to obtain all locks needed before it can execute – requires locks to be obtained in succession

# Optimistic Method

- Most database operations do not conflict
- No locking or time stamping
- Transactions execute until commit
  - Read Phase – Read database, execute computations, make local updates (temporary update file)
  - Validate Phase – Transaction is validated to ensure changes will not effect integrity of database
    - If Validated → Go to Write Phase
    - If Not Validated → Restart Transaction and discard initial changes
  - Write Phase – Commit Changes to database
- Good for Read / Query Databases (Few Updates)

# Database Recovery

- Restore a database from a given state to a previous consistent state
- Atomic Transaction Property (All or None)
- Backup Levels:
  - Full Backup
  - Differential Backup
  - Transaction Log Backup
- Database / System Failures:
  - Software (O.S., DBMS, Application Programs, Viruses)
  - Hardware (Memory Chips, Disk Crashes, Bad Sectors)
  - Programming Exemption (Application Program rollbacks)
  - Transaction (Aborting transactions due to deadlock detection)
  - External (Fire, Flood, etc)

# Transaction Recovery

- Recover Database by using data in the Transaction Log
- Write-Ahead-Log – Transaction logs need to be written before any database data is updated
- Redundant Transaction Logs – Several copies of log on different devices
- Database Buffers – Buffers are used to increase processing time on updates instead of accessing data on disk
- Database Checkpoints – Process of writing all updated buffers to disk → While this is taking place, all other requests are not executes
  - Scheduled several times per hour
  - Checkpoints are registered in the transaction log

# Database Backup

- Backups are useful primarily for two purposes. The first is to restore a state following a disaster (called disaster recovery).
- The second is to restore small numbers of files after they have been accidentally deleted or corrupted.
- Data loss is also very common. 66% of internet users have suffered from serious data loss.