GALGOTIAS UNIVERSITY

School of Computing Science and Engineering

Program: B.Tech CSE

Course Code: BCSE2073

Course Name: Database Management System

Course Outcomes :

CO Number	Title CO
CO1	Explain Database Architecture and Representation Models
CO2	Use DDL and DML commands using SQL to retrieve data from the given table
CO3	Use Normalization techniques to design a database for a given application
CO4	Describe the transaction processing concept and apply storage techniques
CO5	Describe the concurrency control process and various relevant protocols

Course Prerequisites

- ✓ **Basic knowledge of data.**

Syllabus

Unit I: Introduction 9 lecture hours

Introduction: An overview of database management system, database system Vs file system, Database system concept and architecture, data model schema and instances, data independence and database language and interfaces, data definitions language, DML, Overall Database Structure.

Data modeling using the Entity Relationship Model: ER model concepts, notation for ER diagram, mapping constraints, keys, Concepts of Super Key, candidate key, primary key, Generalization, aggregation, reduction of an ER diagrams to tables, extended ER model, relationship of higher degree.

Unit II: Relational data Model and Language 11 lecture hours

Relational data model concepts, integrity constraints, entity integrity, referential integrity, Keys constraints, Domain constraints, relational algebra, relational calculus, tuple and domain calculus.

Introduction on SQL: Characteristics of SQL, advantage of SQL. SQl data type and literals. Types of SQL commands. SQL operators and their procedure. Tables, views and indexes. Queries and sub queries. Aggregate functions. Insert, update and delete operations, Joins, Unions, Intersection, Minus, Cursors, Triggers, Procedures in SQL/PL SQL

Unit III: Data Base Design & Normalization 8 lecture hours

Functional dependencies, normal forms, first, second, third normal forms, BCNF, inclusion dependence, loss less join decompositions, normalization using FD, MVD, and JDs, alternative approaches to database design.

Unit IV: Transaction Processing Concept 8 lecture hours

Transaction system, Testing of serializability, serializability of schedules, conflict & view serializable schedule, recoverability, Recovery from transaction failures, log based recovery, checkpoints, deadlock handling. Distributed Database: distributed data storage, concurrency control, directory system.

Unit V: Concrrency Control Techniques 8 lecture hours

Concurrency control, Locking Techniques for concurrency control, Time stamping protocols for concurrency control, validation based protocol, multiple granularity, Multi version schemes, Recovery with concurrent transaction, case study of Oracle.

Contents

Unit-5 : Concurrency Control

- Lock-Based Protocols
- Timestamp-Based Protocols
- Validation-Based Protocols
- Multiple Granularity
- Multi-version Schemes
- Recovery with concurrent transaction
- Case study of Oracle.

Concurrency Control

Concurrency Control deals with **interleaved execution** of more than one transaction. We will see what is serializability and how to find whether a schedule is serializable or not.

Lock-Based Protocols

- A lock is a mechanism to control concurrent access to a data item
- Data items can be locked in two modes :
  1. *exclusive (X) mode*. Data item can be both read as well as written. X-lock is requested using **lock-X** instruction.
  2. *shared (S) mode*. Data item can only be read. S-lock is requested using **lock-S** instruction.
- Lock requests are made to concurrency-control manager. Transaction can proceed only after request is granted.

Lock-Based Protocols (Cont.)

- **Lock-compatibility matrix**

	S	X
S	true	false
X	false	false

- A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other transactions
- Any number of transactions can hold shared locks on an item, but if any transaction holds an exclusive on the item no other transaction may hold any lock on the item.
- If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released. The lock is then granted.

Lock-Based Protocols (Cont.)

- Example of a transaction performing locking:

  T_2: **lock-S**(*A*);
  read (*A*);
  unlock(*A*);
  lock-S(*B*);
  read (*B*);
  unlock(*B*);
  display(*A*+*B*)

- Locking as above is not sufficient to guarantee serializability - if *A* and *B* get updated in-between the read of *A* and *B*, the displayed sum would be wrong.
- A **locking protocol** is a set of rules followed by all transactions while requesting and releasing locks. Locking protocols restrict the set of possible schedules.

Pitfalls of Lock-Based Protocols

- Consider the partial schedule
- Neither T_3 nor T_4 can make progress — executing **lock-S**(*B*) causes T_4 to wait for T_3 to release its lock on *B*, while executing **lock-X**(*A*) causes T_3 to wait for T_4 to release its lock on *A*.
- Such a situation is called a **deadlock**.
  - To handle a deadlock one of T_3 or T_4 must be rolled back and its locks released.

T_3	T_4
lock-X(B)	
read(B)	
$B := B - 50$	
write(B)	
	lock-S(A)
	read(A)
	lock-S(B)
lock-X(A)	

Pitfalls of Lock-Based Protocols (Cont.)

- The potential for deadlock exists in most locking protocols. Deadlocks are a necessary evil.
- **Starvation** is also possible if concurrency control manager is badly designed. For example:
  - A transaction may be waiting for an X-lock on an item, while a sequence of other transactions request and are granted an S-lock on the same item.
  - The same transaction is repeatedly rolled back due to deadlocks.
- Concurrency control manager can be designed to prevent starvation.

The Two-Phase Locking Protocol

- This is a protocol which ensures conflict-serializable schedules.
- Phase 1: Growing Phase
  - transaction may obtain locks
  - transaction may not release locks
- Phase 2: Shrinking Phase
  - transaction may release locks
  - transaction may not obtain locks
- The protocol assures serializability. It can be proven that the transactions can be serialized in the order of their **lock points** (i.e. the point where a transaction acquired its final lock).

The Two-Phase Locking Protocol (Cont.)

- Two-phase locking *does not* ensure freedom from deadlocks
- Cascading roll-back is possible under two-phase locking. To avoid this, follow a modified protocol called **strict two-phase locking**. Here a transaction must hold all its exclusive locks till it commits/aborts.
- **Rigorous two-phase locking** is even stricter: here *all* locks are held till commit/abort. In this protocol transactions can be serialized in the order in which they commit.

T_3	T_4
lock-X(B)	
read(B)	
$B := B - 50$	
write(B)	
	lock-S(A)
	read(A)
	lock-S(B)
lock-X(A)	

T_5	T_6	T_7
lock-X(A)		
read(A)		
lock-S(B)		
read(B)		
write(A)		
unlock(A)		
	lock-X(A)	
	read(A)	
	write(A)	
	unlock(A)	
		lock-S(A)
		read(A)

The Two-Phase Locking Protocol (Cont.)

- There can be conflict serializable schedules that cannot be obtained if two-phase locking is used.
- However, in the absence of extra information (e.g., ordering of access to data), two-phase locking is needed for conflict serializability in the following sense:
  - Given a transaction T_i that does not follow two-phase locking, we can find a transaction T_j that uses two-phase locking, and a schedule for T_i and T_j that is not conflict serializable.

Refinement of The Two-Phase Locking Protocol:

Lock Conversions

- Two-phase locking with lock conversions:
  - First Phase:
    - can acquire a **lock-S** on item
    - can acquire a **lock-X** on item
    - can convert a **lock-S** to a **lock-X** (**upgrade**)
  - Second Phase:
    - can release a **lock-S**
    - can release a **lock-X**
    - can convert a **lock-X** to a **lock-S (downgrade**)
- This protocol assures serializability. But still relies on the programmer to insert the various locking instructions.

The Two-Phase Locking Protocol:
Automatic Acquisition of Locks

- A transaction T_i issues the standard read/write instruction, without explicit locking calls.
- The operation **read**(D) is processed as:

```
if Ti has a lock on D
  then
      read(D)
  else
      begin
        if necessary wait until no other
           transaction has a lock-X on D
        grant Ti a  lock-S on D;
        read(D)
      end
```

The Two-Phase Locking Protocol: Automatic Acquisition of Locks (Cont.)

- **write**(*D*) is processed as:

```
if Ti has a lock-X on D
   write(D)
else
    begin
       if necessary wait until no other trans. has any lock on D,
       if Ti has a lock-S on D
            then
               upgrade lock on D to lock-X
           else
               grant Ti a lock-X on D
           write(D)
    end;
```

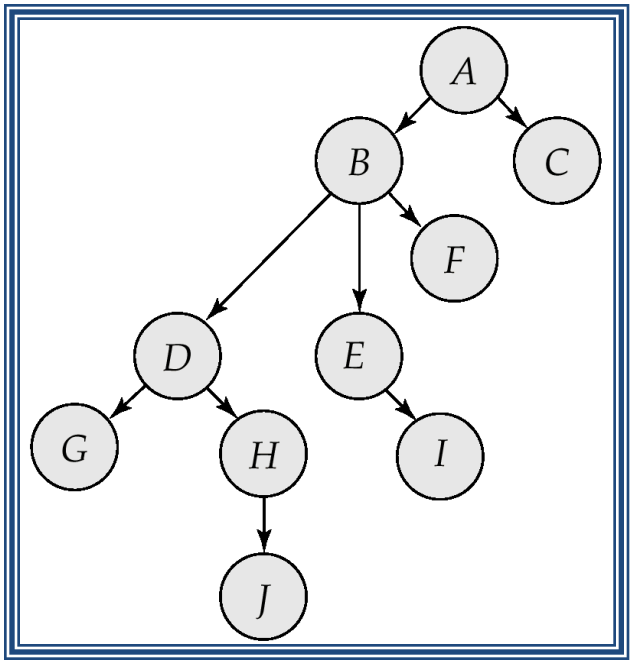
- All locks are released after commit or abort

Graph-Based Protocols

- Graph-based protocols are an alternative to two-phase locking
- Impose a partial ordering $\rightarrow$ on the set **D** = $\{d_1, d_2, ..., d_h\}$ of all data items.
  - If $d_i \rightarrow d_j$ then any transaction accessing both d_i and d_j must access d_i before accessing d_j.
  - Implies that the set **D** may now be viewed as a directed acyclic graph, called a *database graph*.
- The *tree-protocol* is a simple kind of graph protocol.

Tree Protocol

- Only exclusive locks are allowed.
- The first lock by T_i may be on any data item. Subsequently, a data Q can be locked by T_i only if the parent of Q is currently locked by T_i.
- Data items may be unlocked at any time.



Serializable Schedule Under the Tree Protocol

T_{10}	T_{11}	T_{12}	T_{13}
lock-X(B)			
	lock-X(D)		
	lock-X(H)		
	unlock(D)		
lock-X(E)			
lock-X(D)			
unlock(B)			
unlock(E)			
		lock-X(B)	
		lock-X(E)	
	unlock(H)		
lock-X(G)			
unlock(D)			
			lock-X(D)
			lock-X(H)
			unlock(D)
			unlock(H)
		unlock(E)	
		unlock(B)	
unlock (G)			

Graph-Based Protocols (Cont.)

- The tree protocol ensures conflict serializability as well as freedom from deadlock.
- Unlocking may occur earlier in the tree-locking protocol than in the two-phase locking protocol.
  - shorter waiting times, and increase in concurrency
  - protocol is deadlock-free, no rollbacks are required
  - the abort of a transaction can still lead to cascading rollbacks.
    (this correction has to be made in the book also.)
- However, in the tree-locking protocol, a transaction may have to lock data items that it does not access.
  - increased locking overhead, and additional waiting time
  - potential decrease in concurrency
- Schedules not possible under two-phase locking are possible under tree protocol, and vice versa.

Recommended Books

Text books

"Data base System Concepts", Silberschatz, Korth, McGraw Hill, V edition

Reference Book

1. C.J. Date, "An Introduction to Database Systems", Addision Wesley, Eigth Edition, 2003.
2. Elmasri, Navathe, " Fudamentals of Database Systems", Addision Wesley, Sixth Edition, 2011.

Additional online materials

1. WWW.Tutoiralpoint.com/


Thank You