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.

Timestamp-Based Protocols

- An alternative to locking protocols
- Each transaction is issued a timestamp when it enters the system. If an old transaction T_i has time-stamp TS(T_i), a new transaction T_j is assigned time-stamp TS(T_j) such that TS(T_i) <TS(T_j).
- The protocol manages concurrent execution such that the time-stamps determine the serializability order.
- In order to assure such behavior, the protocol maintains for each data Q two timestamp values:
  - **W-timestamp**(Q) is the largest time-stamp of any transaction that executed **write**(Q) successfully.
  - **R-timestamp**(Q) is the largest time-stamp of any transaction that executed **read**(Q) successfully.

Timestamp-Based Protocols (Cont.)

- The timestamp ordering protocol ensures that any conflicting **read** and **write** operations are executed in timestamp order.
- Suppose a transaction T_i issues a **read**(Q)

1. If TS(T_i) ≤ **W**-timestamp(Q), then T_i needs to read a value of Q

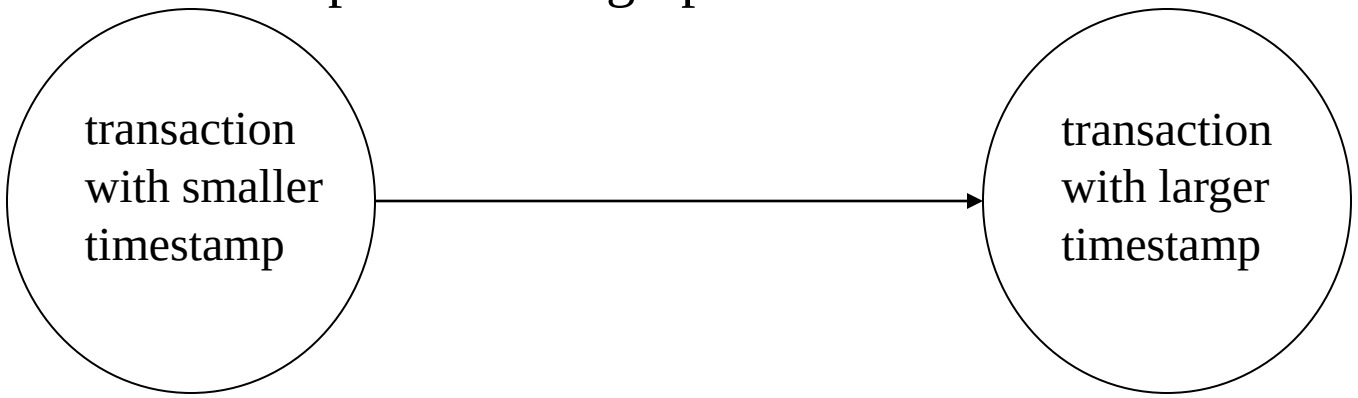
   that was already overwritten. Hence, the **read** operation is rejected, and T_i is rolled back.
2. If TS(T_i)≥ **W**-timestamp(Q), then the **read** operation is executed, and R-timestamp(Q) is set to the maximum of R-timestamp(Q) and TS(T_i).

Timestamp-Based Protocols (Cont.)

- Suppose that transaction T_i issues **write**(Q).
- If TS(T_i) < R-timestamp(Q), then the value of Q that T_i is producing was needed previously, and the system assumed that that value would never be produced. Hence, the **write** operation is rejected, and T_i is rolled back.
- If TS(T_i) < W-timestamp(Q), then T_i is attempting to write an obsolete value of Q. Hence, this **write** operation is rejected, and T_i is rolled back.
- Otherwise, the **write** operation is executed, and W-timestamp(Q) is set to TS(T_i).

Correctness of Timestamp-Ordering Protocol

- The timestamp-ordering protocol guarantees serializability since all the arcs in the precedence graph are of the form:



Thus, there will be no cycles in the precedence graph

- Timestamp protocol ensures freedom from deadlock as no transaction ever waits.
- But the schedule may not be cascade-free, and may not even be recoverable.

Recoverability and Cascade Freedom

- Problem with timestamp-ordering protocol:
  - Suppose T_i aborts, but T_j has read a data item written by T_i
  - Then T_j must abort; if T_j had been allowed to commit earlier, the schedule is not recoverable.
  - Further, any transaction that has read a data item written by T_j must abort
  - This can lead to cascading rollback --- that is, a chain of rollbacks
- Solution:
  - A transaction is structured such that its writes are all performed at the end of its processing
  - All writes of a transaction form an atomic action; no transaction may execute while a transaction is being written
  - A transaction that aborts is restarted with a new timestamp

Validation-Based Protocol

- Useful if majority of transactions are **read-only**
- Execution of transaction T_i is done in three phases.
  1. **Read and execution phase**: Transaction T_i writes only to temporary local variables
  2. **Validation phase**: Transaction T_i performs a "validation test" to determine if local variables can be written without violating serializability.
  3. **Write phase**: If T_i is validated, the updates are applied to the database; otherwise, T_i is rolled back.
- The three phases of concurrently executing transactions can be interleaved, but each transaction must go through the three phases in that order.
- Also called as **optimistic concurrency control** since transaction executes fully in the hope that all will go well during validation

Validation-Based Protocol (Cont.)

- Each transaction T_i has 3 timestamps
  - **Start**(T_i) : the time when T_i started its execution
  - **Validation**(T_i): the time when T_i entered its validation phase
  - **Finish**(T_i) : the time when T_i finished its write phase
- Serializability order is determined by timestamp given at validation time, to increase concurrency. Thus $TS(T_i)$ is given the value of **Validation**(T_i).
- This protocol is useful and gives greater degree of concurrency if probability of conflicts is low. That is because the serializability order is not pre-decided and relatively less transactions will have to be rolled back.

Validation Test for Transaction T_j

- If for all T_i with TS (T_i) < TS (T_j) either one of the following condition holds:
  - **finish**(T_i) < **start**(T_j)
  - **start**(T_j) < **finish**(T_i) < **validation**(T_j) **and** the set of data items written by T_i does not intersect with the set of data items read by T_j.

  then validation succeeds and T_j can be committed. Otherwise, validation fails and T_j is aborted.
- *Justification*: Either first condition is satisfied, and there is no overlapped execution, or second condition is satisfied and

1. the writes of T_j do not affect reads of T_i since they occur after T_i has finished its reads.
2. the writes of T_i do not affect reads of T_j since T_j does not read any item written by T_i.

Schedule Produced by Validation

Example of schedule produced using validation

T_{14}	T_{15}
read(B)	
	read(B)
	B:- B-50
	read(A)
	A:- A+50
read(A)	
(*validate*)	
display (A+B)	
	(*validate*)
	write (B)
	write (A)

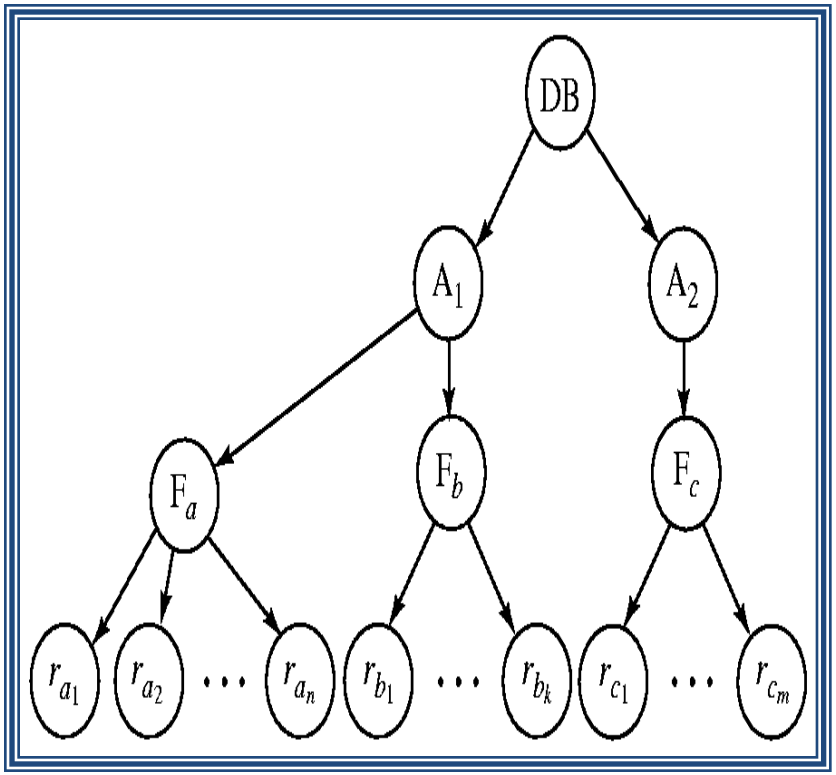
Multiple Granularity

- Allow data items to be of various sizes and define a hierarchy of data granularities, where the small granularities are nested within larger ones
- Can be represented graphically as a tree (but don't confuse with tree-locking protocol)
- When a transaction locks a node in the tree *explicitly*, it *implicitly* locks all the node's descendents in the same mode.
- Granularity of locking (level in tree where locking is done):
  - *fine granularity* (lower in tree): high concurrency, high locking overhead
  - *coarse granularity* (higher in tree): low locking overhead, low concurrency

Example of Granularity Hierarchy

The highest level in the example hierarchy is the entire database.

The levels below are of type *area*, *file* and *record* in that order.



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