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

- ✓ Recoverability, Recovery from transaction failures
- ✓ Testing of Serializability
- ✓ Log based recovery, Checkpoints
- ✓ Deadlock handling
- ✓ Distributed Database: distributed data storage, concurrency control, directory system.

Recoverability

T_8	T_9
read(A)	
write(A)	
	read(A)
read(B)	

- Need to address the effect of transaction failures on concurrently running transactions.
- **Recoverable schedule** — if a transaction T_j reads a data items previously written by a transaction T_i , the commit operation of T_i appears before the commit operation of T_j.
- The following schedule (Schedule 11) is not recoverable if T_9 commits immediately after the read
- If T_8 should abort, T_9 would have read (and possibly shown to the user) an inconsistent database state. Hence database must ensure that schedules are recoverable.

Recoverability (Cont.)

- **Cascading rollback** – a single transaction failure leads to a series of transaction rollbacks. Consider the following schedule where none of the transactions has yet committed (so the schedule is recoverable)

T_{10}	T_{11}	T_{12}
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	
		read(A)

- If T_{10} fails, T_{11} and T_{12} must also be rolled back.
- Can lead to the undoing of a significant amount of work

Recoverability (Cont.)

- **Cascadeless schedules** — cascading rollbacks cannot occur; for each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before the read operation of T_j.
- Every cascadeless schedule is also recoverable
- It is desirable to restrict the schedules to those that are cascadeless

Log-based Recovery

- Log is a sequence of records, which maintains the records of actions performed by a transaction. It is important that the logs are written prior to the actual modification and stored on a stable storage media, which is failsafe.

Log-based recovery works as follows −

- The log file is kept on a stable storage media.
- When a transaction enters the system and starts execution, it writes a log about it.
- <T_n, Start> When the transaction modifies an item X, it write logs as follows −
- <T_n, X, V_1, V_2> It reads T_n has changed the value of X, from V_1 to V_2.
- When the transaction finishes, it logs −
- <T_n, commit>

The database can be modified using two approaches −

- **Deferred database modification** − All logs are written on to the stable storage and the database is updated when a transaction commits.
- **Immediate database modification** − Each log follows an actual database modification. That is, the database is modified immediately after every operation.
- **Recovery with Concurrent Transactions:**

  When more than one transaction are being executed in parallel, the logs are interleaved. At the time of recovery, it would become hard for the recovery system to backtrack all logs, and then start recovering. To ease this situation, most modern DBMS use the concept of 'checkpoints'.

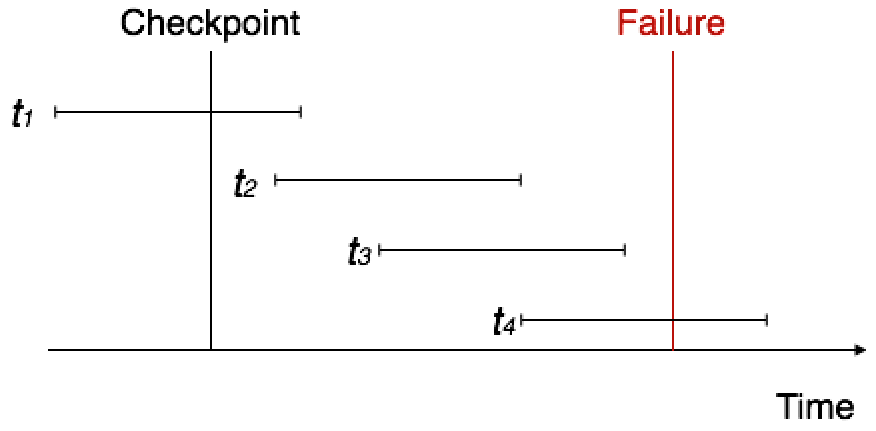
Checkpoint

- Keeping and maintaining logs in real time and in real environment may fill out all the memory space available in the system. As time passes, the log file may grow too big to be handled at all. Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

Recovery

- When a system with concurrent transactions crashes and recovers, it behaves in the following manner-
- The recovery system reads the logs backwards from the end to the last checkpoint.
- It maintains two lists, an **undo-list** and a **redo-list.**

- If the recovery system sees a log with <T_n, Start> and <T_n, Commit> or just <T_n, Commit>, it puts the transaction in the redo-list.
- If the recovery system sees a log with <T_n, Start> but no commit or abort log found, it puts the transaction in undo-list.
- All the transactions in the undo-list are then undone and their logs are removed. All the transactions in the redo-list and their previous logs are removed and then redone before saving their logs.



Implementation of Isolation

- Schedules must be conflict or view serializable, and recoverable, for the sake of database consistency, and preferably cascadeless.
- A policy in which only one transaction can execute at a time generates serial schedules, but provides a poor degree of concurrency..
- Concurrency-control schemes tradeoff between the amount of concurrency they allow and the amount of overhead that they incur.
- Some schemes allow only conflict-serializable schedules to be generated, while others allow view-serializable schedules that are not conflict-serializable.

Transaction Definition in SQL

- Data manipulation language must include a construct for specifying the set of actions that comprise a transaction.
- In SQL, a transaction begins implicitly.
- A transaction in SQL ends by:
  - **Commit work** commits current transaction and begins a new one.
  - **Rollback work** causes current transaction to abort.
- Levels of consistency specified by SQL-92:
  - **Serializable** — default
  - **Repeatable read**
  - **Read committed**
  - **Read uncommitted**

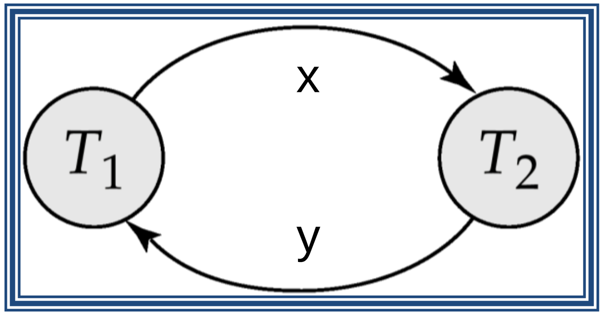
Levels of Consistency in SQL-92

- **Serializable** — default
- **Repeatable read** — only committed records to be read, repeated reads of same record must return same value. However, a transaction may not be serializable – it may find some records inserted by a transaction but not find others.
- **Read committed** — only committed records can be read, but successive reads of record may return different (but committed) values.
- **Read uncommitted** — even uncommitted records may be read.

Lower degrees of consistency useful for gathering approximate information about the database, e.g., statistics for query optimizer

Testing for Serializability

- Consider some schedule of a set of transactions T_1, T_2, ..., T_n
- **Precedence graph** — a direct graph where the vertices are the transactions (names).
- We draw an arc from T_i to T_j if the two transaction conflict, and T_i accessed the data item on which the conflict arose earlier.
- We may label the arc by the item that was accessed.
- **Example 1**



Example Schedule (Schedule A)

T_1	T_2	T_3	T_4	T_5
	read(X)			
read(Y)				
read(Z)				
				read(V)
				read(W)
				read(W)
	read(Y)			
	write(Y)			
		write(Z)		
read(U)				
			read(Y)	
			write(Y)	
			read(Z)	
			write(Z)	
read(U)				
write(U)				

Precedence Graph for Schedule A

$$\begin{tikzcd}
T_1 \arrow[r, bend left] \arrow[d, bend right] & T_2 \arrow[d, bend left] \\
T_3 \arrow[r, bend right] & T_4
\end{tikzcd}$$

Test for Conflict Serializability

- A schedule is conflict serializable if and only if its precedence graph is acyclic.
- Cycle-detection algorithms exist which take order n^2 time, where n is the number of vertices in the graph. (Better algorithms take order $n + e$ where e is the number of edges.)
- If precedence graph is acyclic, the serializability order can be obtained by a ***topological sorting*** of the graph. This is a linear order consistent with the partial order of the graph.
  For example, a serializability order for Schedule A would be
  $T_5 \rightarrow T_1 \rightarrow T_3 \rightarrow T_2 \rightarrow T_4$.

Test for View Serializability

- The precedence graph test for conflict serializability must be modified to apply to a test for view serializability.
- The problem of checking if a schedule is view serializable falls in the class of *NP*-complete problems. Thus existence of an efficient algorithm is unlikely.
  However practical algorithms that just check some *sufficient conditions* for view serializability can still be used.

Concurrency Control vs. Serializability Tests

- Testing a schedule for serializability *after* it has executed is a little too late!
- Goal – to develop concurrency control protocols that will assure serializability. They will generally not examine the precedence graph as it is being created; instead a protocol will impose a discipline that avoids non-seralizable schedules.
  Will study such protocols in Unit-5.
- Tests for serializability help understand why a concurrency control protocol is correct.

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