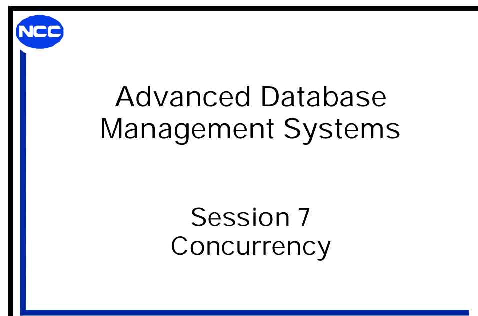
Session 7

Concurrency

1 Introduction

(10 minutes)

V7. 1



Database concurrency and recovery are two inter-related subjects, both being parts of the more general topic of transaction processing.

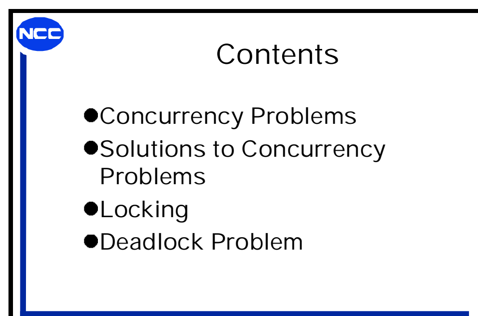
Database systems typically provide multi-user access to a shared database. As such, concurrency control is critical in ensuring that concurrent operations are carried out correctly and efficiently.

This session will discuss the importance of database concurrency, introduce some of its basic ideas and describe various techniques and mechanisms used in the implementation of this function.

Inform students that a handout containing a full set of visuals will be provided to them at the end of this lecture.

1.1 Summary of Topics to be Covered

V7. 2



The topics detailed in the visual will be discussed during this session.

Note: Throughout this session, in addition to the examples provided, further examples and diagrams should be given to enable students to gain a better understanding of the topic.

2 Concurrency Problems

(15 minutes)

V7. 3

Concurrency Problems

There are three concurrency problems:
- The lost update -
  Transaction A's update is overwritten by the subsequent update by transaction B
- The uncommitted dependency -
  Transaction A uses/updates an uncommitted update by B which is rolled back subsequently
- The inconsistent analysis -
  Transaction A uses a data item in an "inconsistent" state (caused by transaction B's access)

There are three concurrency problems, *i.e.* three types of potential mistake which could occur if concurrency control is not properly enforced in the database system.

- *The lost update problem* - This relates to a situation where two concurrent transactions, say A and B, are allowed to update an uncommitted change on the same data item, say x. The second update by transaction B replaces the value of the first update by transaction A. Consequently, the updated value of x by A is lost following the second update by B.
- *The uncommitted dependency problem* - This problem relates to a situation where one transaction A is allowed to retrieve, or update, a data item which has been updated by another transaction B but has not yet been committed by B. Therefore, there is a risk that it may never be committed but be rolled back instead. In that situation, transaction A will have used some data which are now non-existent.

  There are two cases in this category:

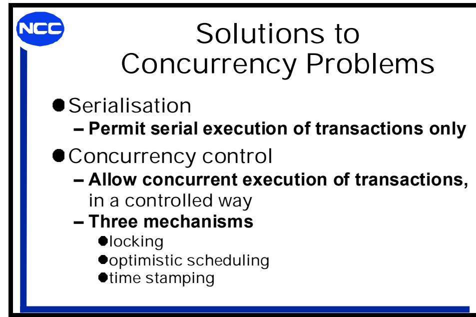
  – Transaction A uses an uncommitted update by transaction B which is subsequently undone.

  – Transaction A updates an uncommitted change by B whose subsequent rollback loses both of the previous updates.
- *The inconsistent analysis problem* - This problem relates to a situation where transaction A uses an data item which is in an *inconsistent state* and as a result carries out an inconsistent analysis.

3 Solutions to Concurrency Problems

(15 minutes)

V7. 4



Various solutions are available to concurrency problems:

- *Serialisation* - One solution is to adopt a policy which permits serial execution of transactions only, *i.e.* transaction A must process a complete transaction before B can start, or *vice versa*.

  The downside of this solution is the slow response time and long CPU idle time.

- *Concurrency control* - Obviously some form of concurrency control mechanism is necessary to enable transactions to run concurrently as far as possible; but controlled in such a way that the effect is the same as if they had been run serially.
- There are three mechanisms available:
  - locking;
  - optimistic scheduling;
  - time stamping.

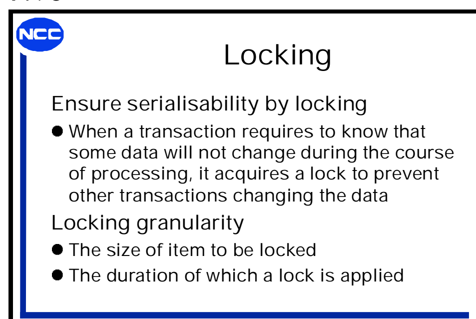
Locking is the most common mechanism of currency control, therefore is concentrated on in greater detail in this session.

It is recommended, however, that an introduction to the basic principle and technique of time stamping is given here.

4 Locking

(40 minutes)

V7. 5



- *Ensure serialisability by locking* - The idea of locking is quite simple: when a transaction requires to know that a data item will not change during the course of processing, it acquires a lock, which locks other transactions out of the item, preventing them changing it. The first transaction knows that the data item remains stable for its duration.

- *Locking granularity* - The size of object to be locked is referred to as the degree of granularity (locking granularity). It could be:
  - the entire database;
  - a disk block;
  - a database relation;
  - a tuple in a relation, or a field of a tuple.

  The larger the data item, the lower is the degree of concurrency, the fewer the number of locks to be set, and the lower the processing overhead.

  The data item should be locked for the whole of the transaction, or to the next synchronisation point.

4.1 Two Types of Lock

V7. 6

NCC

Two Types of Lock

- Exclusive locks (X locks, or write locks)
  - grant read/write access to the transaction which holds the lock
  - prevent any other transactions reading or writing the data
- Shared locks (S locks, or read locks)
  - give read-only access to the transaction which holds the lock
  - prevent any transaction writing the data
  - several transactions may hold a shared lock on an item

There are two types of lock available:

- *Exclusive locks* (X locks, or write locks):
  - grant read/write access to the transaction which holds the lock;
  - prevent any other transactions reading or writing the data.
- *Shared locks* (S locks, or read locks):
  - give read-only access to the transaction which holds the lock;
  - prevent any transaction writing the data;
  - several transactions may hold a shared lock on an item.

4.2 Rules for Locking

V7.7

NCC

Rules for Locking

Compatibility Matrix

B request

A has	X	S	-
X	N	N	Y
S	N	Y	Y
-	Y	Y	Y

Y: grant
N: wait
- : no lock

Locking Protocol
–Get S lock before reading
–Get X lock before writing
–Wait if lock request is denied until its release

- *Compatibility Matrix* - The rules governing the X lock and S lock can be summarised as shown in the compatibility matrix for locks in Visual V7.7.

- *Locking protocol:*
  - A transaction must acquire an S lock on the data item it wishes to retrieve.
  - A transaction must acquire an X lock on the data item it wishes to update.
  - If a lock request is denied, the transaction goes into *wait* state.
  - A transaction in *wait* state resumes operation only when the lock requested is released.
  - X locks are held until Commit or Rollback, S locks are normally the same.

Use examples to demonstrate to the students that the three concurrency problems can be solved by applying the locking technique discussed. Date***'s*** *book contains a discussion on this.*

5 Deadlock Problem

(65 minutes)

Locking can be used to solve the three concurrency problems, but it can also produce the problem of deadlock.

5.1 Definition of Deadlock

"A system is in a state of deadlock if there exists a set of transactions such that every transaction in the set is waiting for another transaction in the set."

In other words, "there exists a set of waiting transactions (T_0, T_1, T_2....T_n) such that T_0 is waiting for a data item which is held by T_1, T_1 is waiting for a data item held by T_2 and T_n is waiting for a data item held by T_0".

V7. 8

NCC

Deadlock Problem

Definition:

"...every transaction in the set is waiting for another transaction in the set"

Example:

Transaction A	Time	Transaction B
-		-
lock p1 Exclusive	t1	-
-	t2	lock p2 Exclusive
lock p2 Exclusive	t3	-
wait	t4	lock p1 Exclusive
wait		wait
wait		wait

Visual V7.8 gives an example of a deadlock situation (Date).

5.2 Deadlock Handling

There are two main methods for dealing with the deadlock problem:

V7. 9

- *Deadlock prevention* - Use a deadlock prevention protocol to ensure that the system will never enter a deadlock state.
- *Deadlock detection and recovery* - Allow the system to enter a deadlock state and then attempt to recover from it using a deadlock detection and recovery scheme.

Both methods may result in transaction rollback. Both methods require overheads.

The prevention method is commonly used if the probability of the system entering a deadlock state is relatively high; otherwise, detection and recovery methods should be used.

5.3 Deadlock Prevention

V7. 10

There are a number of different schemes for deadlock prevention, but the simplest scheme works on the following protocol:

- Each transaction locks all its required data items before it begins execution.
- Either all are locked in one step, or none are locked.

The disadvantages of this scheme are apparent:

- *Low data utilisation* - Many data items may be locked but unused for a long period of time.
- *Possible starvation* - A transaction which requires a number of data items for its operation may find itself in a indefinite wait state while at least one of the data items is always locked by some other transaction.

5.4 Deadlock Detection and Recovery

V7. 11

NCC Deadlock Detection and Recovery

- Basic Idea
  - **Periodically detect whether a deadlock** has occurred in the system
  - **If yes, try to recover from it**
- General Procedure

- *Basic idea* - The state of the system is examined periodically to detect whether a deadlock has occurred. If it has, the system attempts to recover from the deadlock.

- *General Procedure*:
  - Keep information about the current allocation of data items to different transactions, as well as any outstanding requests for data items.
  - Invoke an algorithm which uses this information to determine whether the system has entered a deadlock state.
  - Recover from deadlock.

5.5 Deadlock Detection

V7. 12

NCC Deadlock Detection

- Use wait-for graph
- Detect a deadlock
- Timing for detection

5.5.1 Use Wait-for Graph

Deadlocks can be described by a directed graph called a *wait-for graph.*

- A wait-for graph consists of a pair G = (V,E) where V is a set of vertices and E is a set of edges.
- The set of vertices consists of all transactions in the system.
- Each element in the set E of edges is an ordered pair (Ti, Tj).
- If (Ti, Tj) $\in$ E, then there is a directed edge from transaction Ti to Tj, implying that Ti is waiting for Tj to release a data item it requires.

- When transaction Ti requests a data item currently being held by transaction Tj, then the edge (Ti, Tj) is inserted in the wait-for graph. This edge is removed only when transaction Tj is no longer holding a data item needed by transaction Ti.

5.5.2 Detecting a Deadlock

- A deadlock exists in the system if and only if a cycle is found in the wait-for graph. Each transaction involved in the cycle is then said to be deadlocked.
- To detect deadlocks, the system maintains the wait-for graph, and periodically invokes an algorithm which searches for a cycle in the graph.

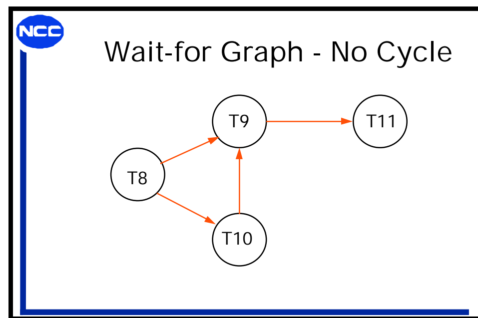
5.5.3 Timing for Detection

The question of when and how often the detection algorithm should be called into action depends mainly on the following factors:

- the frequency of deadlock occurrences;
- the number of transactions which would likely be affected by the deadlock.

5.5.4 Examples

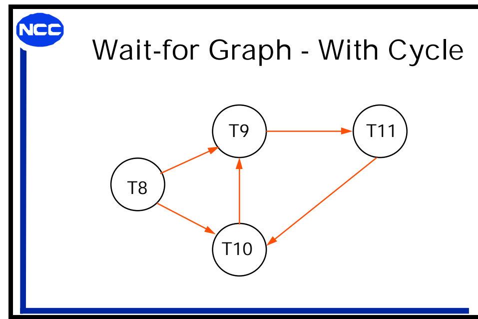
V7. 13



- The example given in Visual V7.13 illustrates a *wait-for* graph with no cycle:
  - Transaction T8 is waiting for transactions T9 and T10.
  - Transaction T10 is waiting for transaction T9.
  - Transaction T9 is waiting for transaction T11.

It is clear that the system is not in a deadlock state, as no cycle is present in the graph.

V7.14

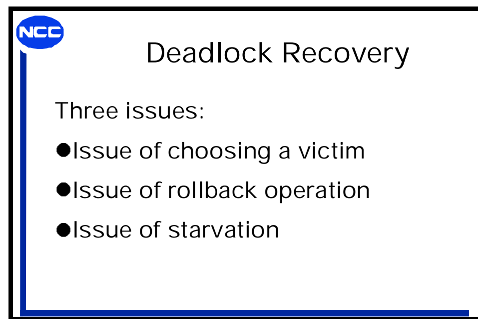


Visual V7.14 gives an example of a *wait-for* graph *with* cycle. If transaction T11 is now requesting an item held by T10, then the edge (T11, T10) will be added to the wait-for graph. As a result, the graph for the new system state contains one cycle:

It can be seen that T9 → T11 → T10 → T9 forms a cycle, implying that transactions T9, T10, and T11 are all deadlocked.

5.6 Deadlock Recovery

V7. 15



- After a detection algorithm has identified a deadlock, the system must try to recover from it.
- The most common solution is to roll back one or more transactions so that the deadlock can be broken.
- Bear in mind that data items held by deadlocked transactions will be unavailable to other transactions until the deadlock is broken.

Three issues need to be addressed in deadlock recovery:

- *Issue of choosing a victim*:
  - Determine which transaction(s), among a set of deadlocked transactions, to roll back to break the deadlock.
  - Criteria for choosing a victim depends mainly on the cost factor: choose one which will incur the minimum cost.
  - The information which assists the calculation of costs includes:
    - how long the transaction has computed, and how much longer the transaction will compute before completing its job;
    - how many data items the transaction has used, and how many more are needed for the transaction to complete;
    - how many transactions will be involved in the rollback.

- *Issue of rollback operation*:
  - Determine how far the chosen victim transaction should be rolled back.
  - Simple solution: total rollback, *i.e.* abort the transaction and then restart it.
  - A more complex solution: roll back the transaction only as far as necessary to break the deadlock.
- *Issue of starvation*:
  - Some transaction may always be chosen as the victim due to cost factors based selection. This may prevent it from ever completing its job.
  - This can be avoided by adopting a simple policy that a transaction can be chosen as a victim only a (small) finite number of times. The most common solution is to include the number of rollbacks in the cost factor.

6 Summary

(5 minutes)

V7. 16

Summary

- Importance of database concurrency
- Three types of concurrency problem
- Solutions to concurrency problems
- Locking mechanism for concurrency control
- Deadlock handling: its cause, prevention, detection and recovery
- Rôle of the wait-for diagram in deadlock detection

Concurrency control in a multi-user database environment is an essential function performed by the underlying database management system. This session has discussed the topic, introduced its concept, principles and some of the basic and important techniques used for database concurrency control. The key points are as follows:

- Importance of database concurrency.
- Three types of concurrency problem.
- Solutions to concurrency problems.
- Locking mechanism for concurrency control.
- Deadlock handling: its cause, prevention, detection and recovery.
- Rôle of the wait-for diagram in deadlock detection.