Schlageter proposed an environment to kung Robinson method whereby a read transaction always proceeds without the risk of restarts, a validation check and thus without the

8.00 Validation Protocol:- In all concurrency control techniques (locking & timestamping), a certain
9.00 degree of checking is done before database operation can be executed. For example, in a locking, a
10.00 check is done to determine whether item being accessed is locked. In timestamp ordering, the transaction
11.00 timestamp is checked against read and write timestamp of the item. Such checking represents overhead during
12.00 transaction execution, with effect of slowing down the transactions and thereby reducing the concurrency.
1.00 According to kung and Robinson; no checking is done while transaction is executed. It is optimistic because
2.00 the likelihood of two clients transactions accessing the same object is low. Transactions are allowed to proceed
3.00 as though there is no possibility of conflict with other transaction until client completes it task and issues close transaction request. When conflict
4.00 arises, some transaction is generally aborted and will need to be restarted by the client.

5.00

As Validation & update phase are generally short in
6.00 duration compared to working phase, only one transaction may be in validation and update phase at one time.

Evening

In validation phase for transaction T_i, the protocol checks that T_i doesn't interfere with any committed transactions or with any other transactions currently in their validation phase.
SUNDAY 10
The validation phase for T_i checks that for each such transaction T_j that is either committed or is in its validation phase, one of the following conditions holds:

Notes

1) Transaction T_j completes its write phase before T_i starts its read phase. Finish (T_j) < Start (T_i)

2) Set of data items written by T_j doesn't intersect with set of data items read by T_i and T_j completes its write phase before T_i starts its validation phase. Write of T_j

6.00

There are 3 phases in validation protocol;⇒

1) Read phase:→ Transaction read values ~~from~~ of committed data items from database. Updates are applied only to local copies of data items.

2) Validation phase:→ Checking is performed to ensure serializability will not be violated if the transaction updates applied to database

3) Write phase:→ If validation phase is successful, updates are applied to database otherwise updates are discarded and transaction restarted

The idea behind this technique is to do all checks at once; hence transaction execution proceeds with minimum overhead until validation phase is reached.

donot overlap.

Multiversion timestamp:→ It was introduced by Reed in 1983. A list of old committed versions as well as tentative versions is kept for each object. This list represents the history of the values of the object. The benefit of using multiple versions is that read operations that arrived too late need not be rejected.

When a transaction requires access to an item an appropriate version is chosen to maintain serializability of the currently executing schedule if possible.

An obvious drawback of this technique is that more storage is needed to maintain multiple versions of the database items. However, older versions may have to be maintained anyway — for example for recovery purpose. They are mainly used in temporal database which keeps track of all changes and the time at which they occured.

Each version has following fields:→

i) Content of X.
ii) read timestamp :→ Largest of all timestamps of transactim that have successfully read version X.
iii) write timestamp :→ Timestamp of transactim that wrote value of X.

Whenever a transaction T_i is allowed to execute a write operation, a new version X_{k+1} of item X is created, (with timestamp T) with readtimestamp and writetimestamp set to TS(T). Correspondingly, when a transaction T is allowed to read the value of version X_i, the read timestamp is set to

when a version X_i has write time t such that blue transaction has a timestamp less than t, then we may delete version of X previous to X_i.

larger of current read timestamp and TS(T).

To ensure serializability, following rules are used.

a) If Transaction T issues read(x), find the version of X that has highest write timestamp of all versions of X that is equal or less than TS(T) and ~~read TS(Xi) > TS(T)~~ then return the value of X_i to transaction T, and set the value of read TS(X_i) to larger of TS(T) and current read TS(X_i).

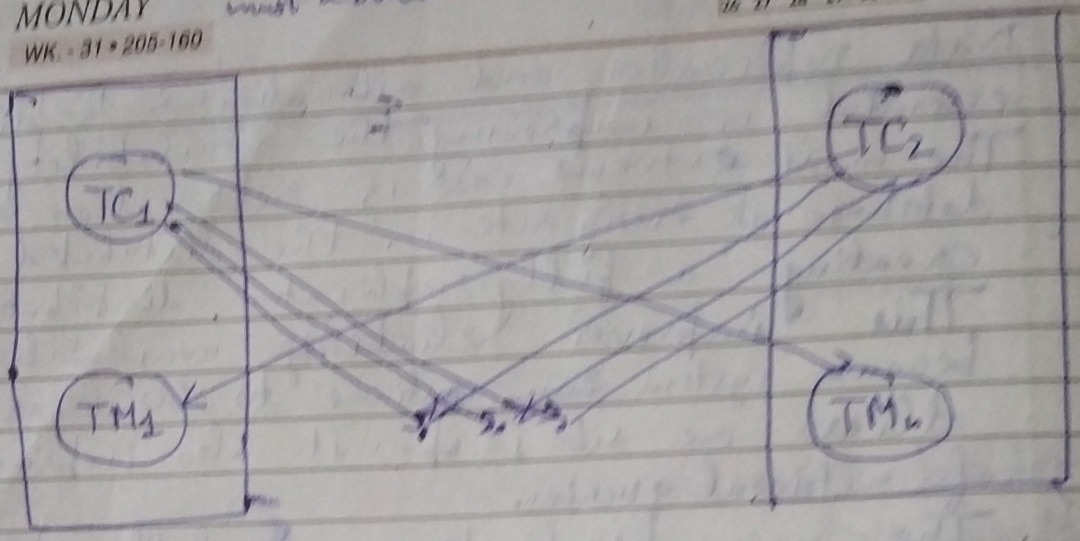
b) If transaction T issues write(x) and TS(T) < Read timestamp (X_i), then abort T other create a new version of X. with r(T)=w(T)=TS(T).

~~It has~~

~~X~~ Multiversion 2 Phase locking → This protocol differentiates read only transactions and update transactions. Update transactions perform rigorous 2 PL

In multiversion 2 PL, we ~~perform~~ allow other transactions T to read an item X while a single transaction T holds a write lock on X. This is accomplished by allowing two versions for each item X; one version must have always been by some committed transaction The second version X' is created when a transaction acquires write lock on item. Other transactions can continue to read the committed version of X while T holds write lock.

Once T is ready to commit, it must obtain certify lock on all items that it currently holds write locks on before it can commit. All read locks on these items be released in order to ~~by~~ acquired Once certified lock of X is set to value of version X'.



TC is responsible for
a) Starting execution of the transaction.
b) Breaking transactions into a number of subtransactions and distributing these subtransactions to appropriate sites for execution.
c) Coordinating the ~~termination~~ outcome of the transaction, which may result in the transaction being committed at all sites or aborted at all sites.

Basic failure of ddbms: ⇒	Solution: ⇒
a) Failure of site	
b) Loss of message	Retransmission,
c) Failure of communication link	
d) N/w Partition	

Two phase Commit protocol: ⇒

Goal of 1st phase ⇒ To reach a common decision
" " 2nd " ⇒ To implement the decision.

~~1st phase~~ The coordinator is responsible for taking the final commit or abort decision. Each participant

corresponds to a subtransaction which has performed some write action i.e. it is responsible for performing write action at its local database.

Phase I:→ Here, the coordinator asks all the participants to prepare for commitment; each participant answers Ready if it is ready to commit. Before sending the message, the coordinator records on stable storage a log record of <prepare T>. On receiving such a message, the transaction mgr at that site determines whether it is willing to commit its portion of T. If the answer is no, it adds record <no T> to the log and then responds by sending abort T message to C_i.

If the answer is yes, it adds a record <ready T> to the log and forces the log onto stable storage. Then the TM then replies with a ready T message to C_i.

Phase II:→ The coordinator decides whether to commit or abort as a result of the answer it received from participants. If all the participants answer ready, it decides to commit. If some participant has answered Abort or has not yet answered when the timeout expires it decides to abort the transaction. Depending on verdict, the fate of the transaction has been sealed. either a record <commit T> or a record <abort T> is added to the log. Afterwards, the coordinator sends either a commit T or abort T message to all participants. When site receive message it records the message in the log.

Finally, all participants send a final acknowledgement message to the coordinator and perform the actions required for committing or aborting subtransactions. When the coordinator has received an Ack message from all participants it writes a log record called

JULY 2006

M	T	W	T	F	S	S
31					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

AUGUST 2006

M	T	W	T	F	S	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

JUNE 2006

THURSDAY 08

WK. - 24 • 159-206

8.00

Multiple Granularity :→ A database item could

a) a record ✓

9.00 b) a field value ✓

c) disk block ✓

10.00 d) whole file ✓

e) whole database ✓

11.00

The granularity can affect the performance of concurrency control and recovery. The size of data items is known as data item granularity. Fine granularity refers to small item sizes whereas coarse granularity refers to large item sizes. Larger the data item size, lower is the degree of concurrency permitted.

12.00 Smaller the size, more is the no. of active locks causing higher overhead. Also, more storage space required for lock table.

1.00 So, what is the best item size? Well, it depends on the types of transactions involved.

2.00 Case:

a) If transaction accesses a small no. of records, then we can make single record as data item.

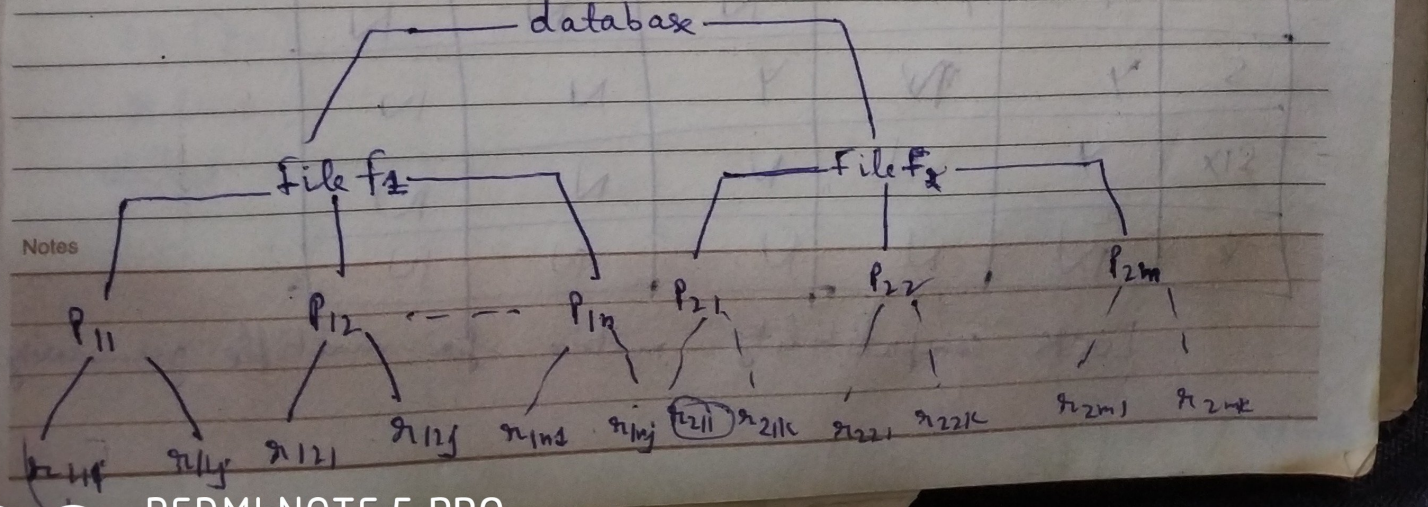
3.00 b) If transaction involves many records in the same file, it may better to have block or file granularity.

4.00

5.00

6.00

Evening



Notes

2006 JUNE

09 FRIDAY

WK - 24 • 160-205

The multiple granularity protocol requires that lock be acquired in top down (root to leaf) order, whereas locks must be released in bottom-up (leaf-to-root) order.

Rules :—

1) Adhere to lock compatibility matrix.
2) Root of tree must be locked first and can be locked in any mode.
3) A node Q can be locked by T_i in S or IS mode only if the parent of Q is currently locked by T_i in either IX or IS mode.
4) A node Q can be locked by T_i in X, SIX or SIX only if the parent of Q is currently locked by T_i in either IX or SIX mode.
5) T_i can lock a node only if it has not previously unlocked any node i.e. T_i is two-phase.
6) T_i can unlock a node Q if none of children of Q are currently locked by T_i.

	IS	IX	S	SIX	X
IS	Y	Y	Y	Y	No
IX	Y	Y	N	N	N
S	Y	N	Y	N	N
SIX	Y	N	N	N	N
X	N	N	N	N	N

Lock compatibility matrix for multiple granularity

JUNE 2020

SATURDAY 10

The idea behind intention locks is for a transaction to indicate, along the path from root to the desired node, what type of lock (shared or exclusive) it will require of one of the node's descendants.

- IS → shared lock will be requested on some descendant node.
- IX → exclusive lock will be requested on some descendant node.
- SIX → current node is locked in shared mode but exclusive lock will be requested on some descendant node

a) Update record r_{111} & r_{211} (b) update records on page p_{12} (c) records in other file