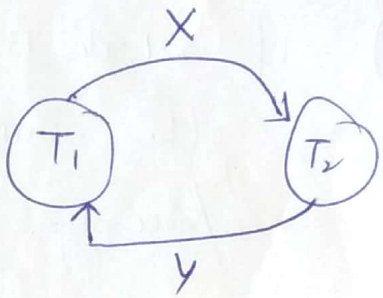
Dealing with Deadlocks

Deadlock: It occurs when each transaction T in a set of two or more transactions is waiting for some item that is locked by some other transaction T' in the set.

eg:

T_1	T_2
lock(Y)	
R(Y)	
	lock(X)
	R(X)
W(X)	
	W(Y)

Cause deadlock.



Deadlock Prevention:

→ To prevent deadlock, use deadlock prevention protocol.

→ Two schemes that prevent deadlock are called wait-die & wound-wait.

→ Suppose that transaction T_i tries to lock an item X but is not able to because X is locked by some other transaction T_j with a conflicting lock. The rules followed by these schemes are as follows:-

Thomas's Write Rule:

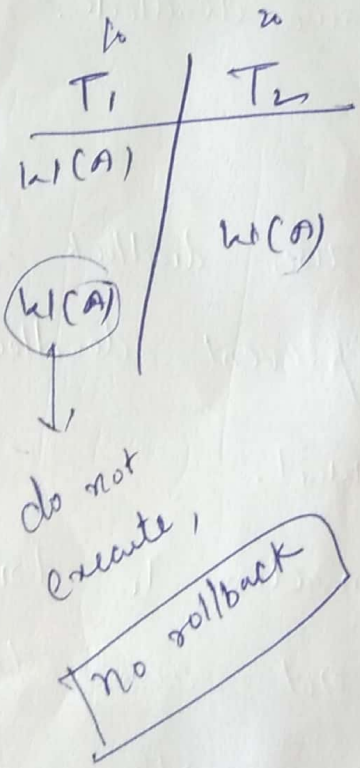
Modification of basic TO by modifying the check for the write_item(X) operation as follows:-

(1) if read_TS(X) > TS(T), then abort & rollback T & reject the operation.

(2) if write_TS(X) > TS(T), then do not execute the write operation but continue processing.

(3) else execute the write_item(X) & set write_TS(X) = TS(T)

eg:



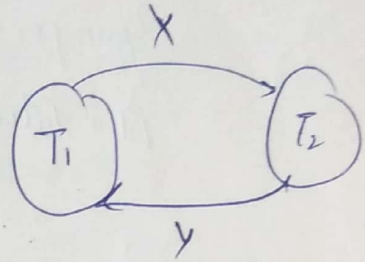
R TS(A) = 0
W TS(A) = ∅ / 20

Deadlock

→ It occurs when each transaction T_i in a set of two or more transactions is waiting for some item that is locked by some other transaction T_j in the set.

eg :→

T_1	T_2
read_lock(Y);	
read_item(Y);	
	read_lock(X);
	read_item(X);
write_lock(X);	
	write_lock(Y);



cause deadlock.

Deadlock Prevention Protocols

→ One way to prevent deadlock, use deadlock prevention protocol.

→ Two schemes that prevent deadlock are called wait-die & wound-wait.

→ Suppose that the transaction T_i tries to lock an item X but is not able to because X is locked by some other transaction T_j with a conflicting lock. The rules followed by these schemes are as follows:

1) Wait-die : It is a non-preemptive technique.

i) If $TS(T_i) < TS(T_j)$ that means T_i which is requesting a conflicting lock, is older than T_j, then T_i is allowed to wait.

ii) If $TS(T_i) > TS(T_j)$ that means T_i is younger than T_j → T_i dies and restart it later with the same timestamp.

(2) ~~Hours~~

5	10	15
T_1	T_2	T_3
	lock(X) R(X)	
lock(X)		
		lock(X)

If T_1 & T_3 both are requesting for X hold by T_2,

then T_1 has to wait

& T_3 → dies and rollback after some time.

(2) Wound-Wait : It is a preemptive technique.

1) If $TS(T_i) < TS(T_j)$, then T_i forces T_j to be rolled back ie T_i wounds T_j. T_j is restarted later on with same timestamp.

2) If $TS(T_i) > TS(T_j)$ then T_i is forced to wait until the resource is available.

eg:

5	10	15
T_1	T_2	T_3
	lock(x)	
	R(A)	
lock(x);		lock(x);

T_1 & T_3 are requesting for x hold by T_2, then T_1 wounds T_2 & T_2 has to rolled back.

& T_3 wait until database is available.

Deadlock detection / Avoidance :->

→ In this, Check whether the deadlock actually exists or not.

→ Deadlock is detected ~~so that~~ to check

→ ~~A simple~~ whether the deadlock is avoidable or not.

→ A simple way to detect a state of deadlock is for the system to construct and maintain wait-for graph.

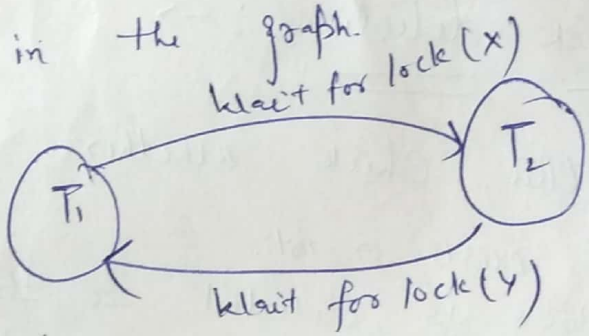
Wait-for-Graph :→

→1) For each transaction entering into the system, a node is created.

→2) When a transaction T_i requests for a lock on an item, say X, which is held by some other transaction T_j, a directed edge is created from T_i to T_j.

3) If T_j releases item X, the edge b/w them is dropped & T_i locks the data item.

4) The system keeps checking if there is any cycle in the graph.



T_1	T_2
~~read~~ write lock(Y)	
	write ~~read~~ lock(X)
write ~~read~~ lock(X)	
	write lock(Y)

deadlock

The following approaches to avoid deadlock are:

1) Do not allow any request for an item, which is already locked by another transaction. This is not always feasible and may cause starvation, where a transaction indefinitely waits for a data item and can never acquire it.

2) Rollback one of the transactions: It is not always feasible to rollback the younger transaction, as it may be important than the older one. With the help of some relative algorithm, a transaction is chosen, which is to be aborted. This transaction is known as Victim & process is known as victim selection.

Algorithm for victim Selection:

It should generally avoid selecting transactions that have been running for a long time and that have performed many updates, and it should try instead to select transactions that have not made any changes.

Introduction to Recovery Techniques :→

→ Recovery is required to protect the database from any data loss and inconsistencies.

→ The recover manager ensures the ~~atomicity~~ ACID ~~& durability~~ properties of a transaction.

Types of failures :→

1) System crashes [eg: Power failure, Software Pbm, Virus etc]
2) System Error [Pbm in Software like syntax error eg: unclesirable state)
   To avoid this, transaction has to rollback.
3) Disk failure
4) logical errors [eg: at a time of execution, correct data not found]
   eg: do not identify identification]
5) Concurrency control enforcement [eg: when concurrent access on database, eg: Lost update problem, Dirty read, Incorrect Summary]
6) Natural & Physical failure [eg: Power failure, Sparkly, etc, earthquake, etc]

→ System crash & logical errors are common type of failures & result in loss of volatile Storage (such as mainmemory, cache) so they are also known as non-catastrophic failures.

→ While, disk failure & failure due to nature result in loss of non-volatile storage & known as catastrophic failure.

Why Recovery is needed

Types of failures :→

1. Computer failure : H/w, S/w, n/w, crash.
2. Transaction or system error
3. local errors or exception condition detected by transaction eg: no balance in account
4. Concurrency control enforcement : deadlock
5. Disk failure
6. Physical problems or Catastrophic failures : power failure, fire, theft, etc.

Recovery Techniques based on Deferred Update :-> (1)

→ The idea behind deferred update is to defer or postpone any actual updates to the database itself until the transaction completes its execution successfully & reaches its commit point.

The steps involved in the deferred update are as follows :-

1. When a transaction starts, write an entry Start_transaction (T) to the log;
2. When any operation is performed ~~that will change values in the database~~, write a log entry write_item (T, X, old_value, new_value)
3. When a transaction is about to commit, write a log record of the form commit (T)
4. Commit the transaction, using the log to write the updates to database.
5. If transaction aborts, ignore the log records.

Hence, there is no need to UNDO Any operations.

∴ This technique is also known as NO-UNDO/REDO Algorithm.

* REDO is needed in case the system fails after the transaction commits but before all its changes are recorded in the database.

(2) Checkpoints :→

→ Another type of entry in the log is called a ch.

→ A [checkpoint] record is written into the log periodically at that point when the system writes out to the database on disk all DBMS buffers that have been modified.

→ All transactions that have their [commit, T] entries in the log before a [checkpoint] entry do not need to have their write operations reclone in case of system crash, since all their updates will be recorded in the database on disk during checkpointing. [further from Navathe Pg- 675]

Recovery using Deferred update in a Single-User Environments :→

The algorithm RDU_S uses a REDO procedure :→

Algorithm RDUS :→

* Uses two list of transactions :

1) Committed transactions since the last checkpoint
2) Active transactions

PROcedure REDO :→

Redoing a write_item operation WRITE_OP consist of examining its log entry and setting the value of item X in the database to new_value.

eg: [Start_transaction, T_1]
[write_item, T_1, D, 20]
[commit, T_1]
[start_transaction, T_2]
[write_item, T_2, B, 10]
[write_item, T_2, D, 25] → system crash

∴ operations of T_1 & redone & T_2 log entries are ignored by recovery process.

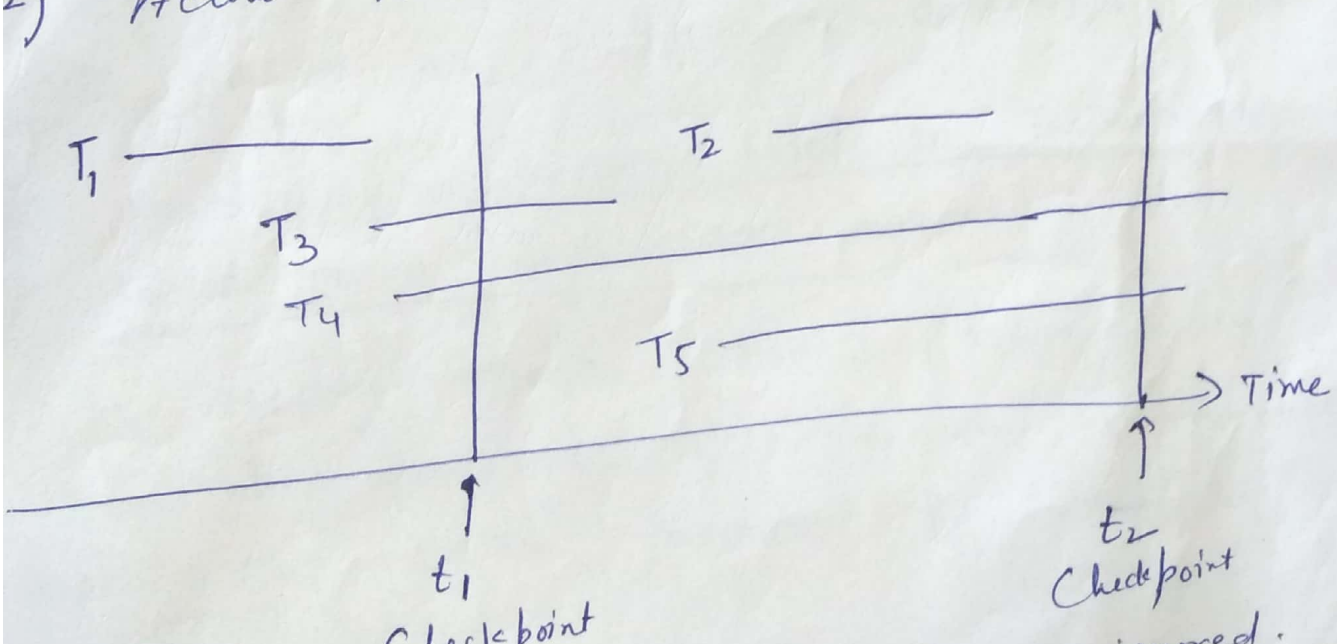
In Multiuser Environment :->

RDU_M uses REDO procedure:

Procedure RDU_M (with checkpoints) :->

Uses two list of transactions:

1) Committed transaction T since the last checkpoint (commit list)
2) Active transactions T' (active list)



do need to redo T_1
T_2 & T_3 are redo after t_2
. T_4 & T_5 are ignored.

is held by some

REDO: Redo all the write operations of the committed transactions from the log, in the order in which they were written into the log. The transactions that are active and did not commit are effectively Canceled and must be resubmitted.

eg:

```
[start_transaction, T1]
[write_item, T1, D1, 20]
[commit, T1]
[checkpoint]
[start_transaction, T4]
[write_item, T4, B, 15]
[commit, T4]
[start_transaction, T2]
[write_item, T2, B, 12]
```

← System Crash

T_2 is ignored

T_4 is redone, T_1 is already Saved.

Recovery Techniques based on Immediate Update :->

→ In these techniques, when a transaction issues an update command, the database can be updated immediately, without any need to wait for the transaction to reach its commit point.

Two main categories of immediate update algorithm are :-

(1) If the recovery technique ensures that all updates of a transaction are recorded in the database on disk before the transaction commits, there is never a need to REDO any operations of committed transactions. This is called as UNDO/NO-REDO recovery algorithm.

(2) UNDO/REDO recovery algorithm:
If transaction is allowed to commit before all its changes are written to database.

UNDO/REDO recovery based on immediate update in a Single-User Environment :->

The recovery algorithm RIU_S (recovery using Immediate update in a Single-user environment) uses the REDO procedure as well as UNDO procedure :

Procedure RIU_S :→ (Pg-683)

1. Use two lists of transactions: committed transactions & active transactions
2. Undo all write_item operations of active transactions.
3. Redo write_item operations of committed transactions.

Procedure UNDO :→ Undoing a write_item operation consists of examining its log entry.

Procedure REDO : Redoing a write_item operation.

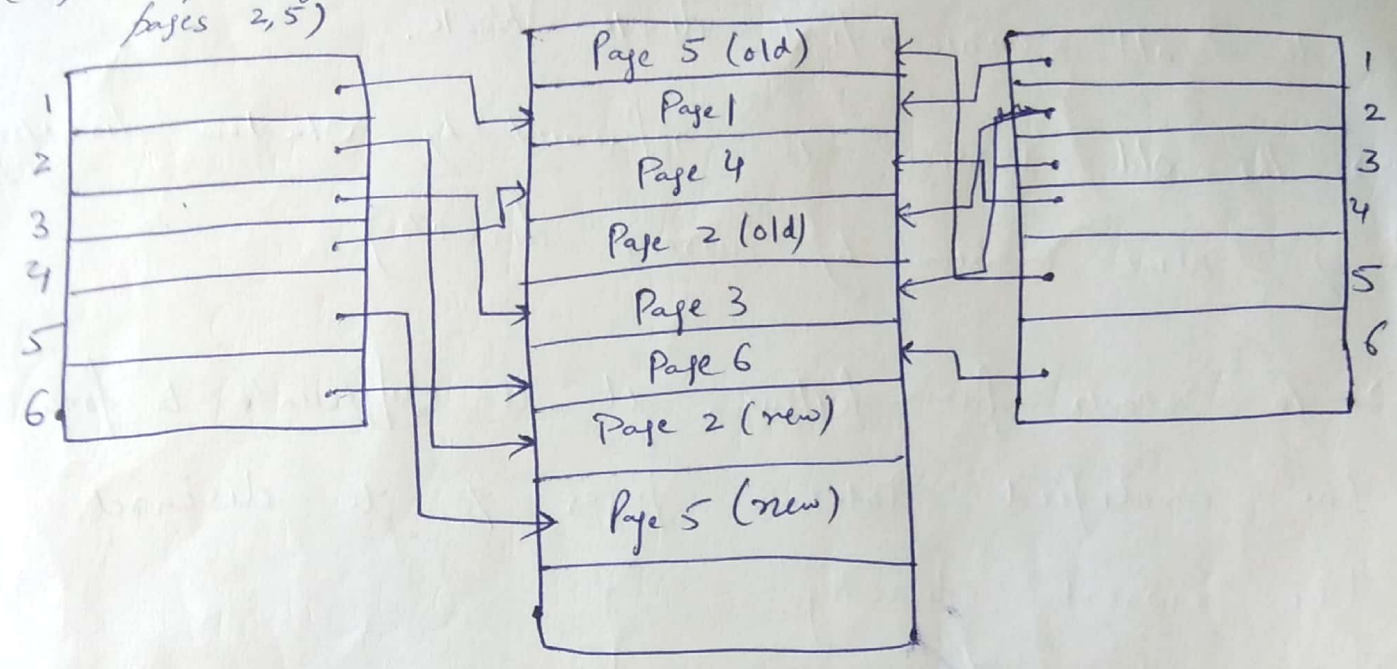
UNDO/REDO recovery based on immediate update with concurrent execution :→

Assume log includes checkpoints and concurrency control protocol produces strict schedules.

Procedure RIU_M :→

1) Use two lists of transactions: committed & active
2) Undo all write_item operations of active transactions
3) Redo write_item operations of committed ".

Shadow Paging :→ (NO-UNDO/NO-REDO)



→ When a transaction begins executing, the current directory-whose entries point to the most recent or current database pages on disk is copied into a shadow directory.

→ The shadow directory is then saved on disk while the current directory is used by transaction.

→ During transaction execution, shadow directory is never modified.

→ When a write_item operation is performed, a new copy of the modified database page is created but the old copy of that page is not overwritten.

→ The current directory entry is modified ~~to continue~~ to point to the ~~old~~ new block, whereas Shadow directory is not modified & continues to point to old unmodified disk block.

→ The old version is referenced by Shadow directory & new version by current directory.

→ To recover from failure, it is sufficient to free the modified database pages & to discard the current directory.

→ The database is returned to state prior to transaction that was executing when the crash occurred & modified pages are discarded.

→ committing a transaction corresponds to discarding the previous Shadow directory.

Introduction to Database Security :→

Database can be managed by 2 different modes of security controls:

(1) Authentication

(2) Authorization

Authentication :-

→ It is the process of confirming that a user logs in only in accordance with the rights to perform the activities he is authorized to perform.

→ User Authentication can be performed at operating system level or database level itself.

→ for authentication, it requires two different credintials, those are userid & password.

Authorization :

→ It is the process managed by database manager.

→ The manger obtains information about current authenticated user.

Different ways of permissions available for authorization:

1) Primary permission : Grants the authorization ID directly.
2) Secondary " : Grants to groups if user is member.
3) Public " : Grants to all users.
4) Context-Sensitive " : Grants to trusted context a/c.

Discretionary Access control based on Granting and Revoking Privileges :→

DAC: It is used to grant privileges to users including capability to access specific data files or fields in specific mode.

There are two levels for assigning privileges to use the database system. At this level, DBA specifies the privileges that each account hold independently.

In case of DAC, a given user typically has different access rights on different objects.

(1) Account level :- It include previliges like Create Table, Create View, Alter, Drop, Modify, Select, etc.

(2) Relation level :- At this level, DBA can control the previlige to access each individual relation or view in database.

→ The granting and revoking of previliges generally follow an authorization model for discretionary previliges known as Access Matrix Model, where rows of a matrix M represent subjects (users) & colum represent ~~refere~~ ~~relations~~ objects (relations, records).

→ The following types of previliges can be granted on each individual relation R :->

1) select previlege on R
2) modify " " "
3) reference " " R.

→ Revoking previliges : It is included for cancelling previliges.

Privileges :- It represents the right of a particular user to access, create, manipulate & destroy various objects ~~inside~~ inside a database.

It can be granted in two ways:

1) Explicitly by giving right to user
2) By granting privileges to roles & then granting roles to users.

There are two distict categories of privileges:

1) System Privilege : It allow users to perform some administrative tasks within a given DBMS. eg: Create Table, Session
   Create View
   Drop privileges.
   Alter any procedure, role.

2) Object Privilege: are those granted from a user to access to manipulate database objects.
   eg: Select, Insert, Update, Delete.

Ques: →

Teacher_Id	Department	HoD	Percent_Time
100	Computers	Vinod Kumar	50
200	Mathematics	S. Sandhu	60
200	Physics	S. Brijesh	40
300	History	Arman Kumar	30

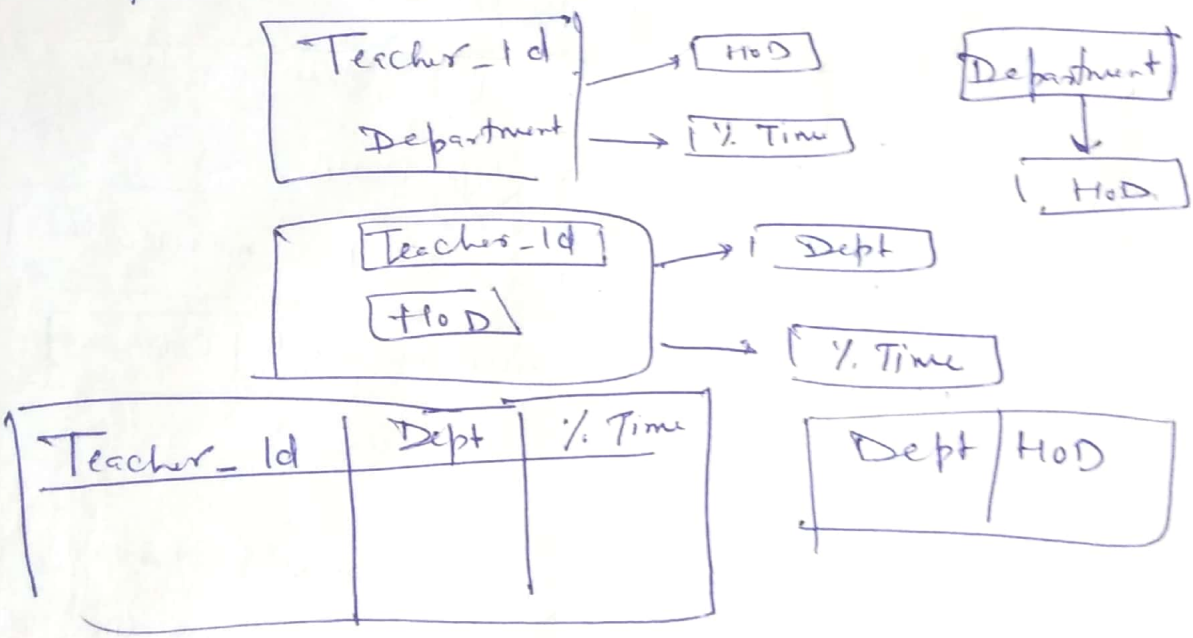
Assume teacher can work in more than 1 dept, % age time he spent in each dept is given but each dept has only 1 HoD.

Ques

2 keys : → 1) (Teacher_Id, Department)

2) (Teacher_Id, HoD)

Dependencies:



Teacher_Id	Dept	% Time

Dept	HoD

Que :->

Client_Interview Relation :->

Client_no	Interview_date	Interview_time	Staff_no	Room_no
CR76	13/5/95	10:30	SG5	G101
CR56	13/5/95	12:00	SG5	G101
CR74	13/5/95	12:00	SG17	G102
CR56	1/7/95	10:30	SG5	G102

Dependencies :->

1) Client_no, Interview_date → Interview_time, Staff_no, Room_no

2) Staff_no, Interview_date → ~~Client_no~~ Room_no

3) Staff_no, Interview_date, Interview_time → Client_no, Room_no

↓↓ BCNF

1) | Client_no | Interview_date | Interview_time | staffno |

2) | Staff_no | Interview_date | Room_no |

Que 1NF :->

Order [Order_no, Order_dt, Cust_name, Cust_addr, Ship_addr, Phone1, Phone2, Cust_disc, Product_code, Prod_desc, Qty, ~~Total~~ Price]

1) 1NF: Order [Order_no, Order_dt, Cust_name, Cust_addr, Ship_addr, Phone1