must use the old value field of the log records described to restore the modified datda items to the value they had prior to the start of the transaction. The undo operation described next, accomplishes this restoration.

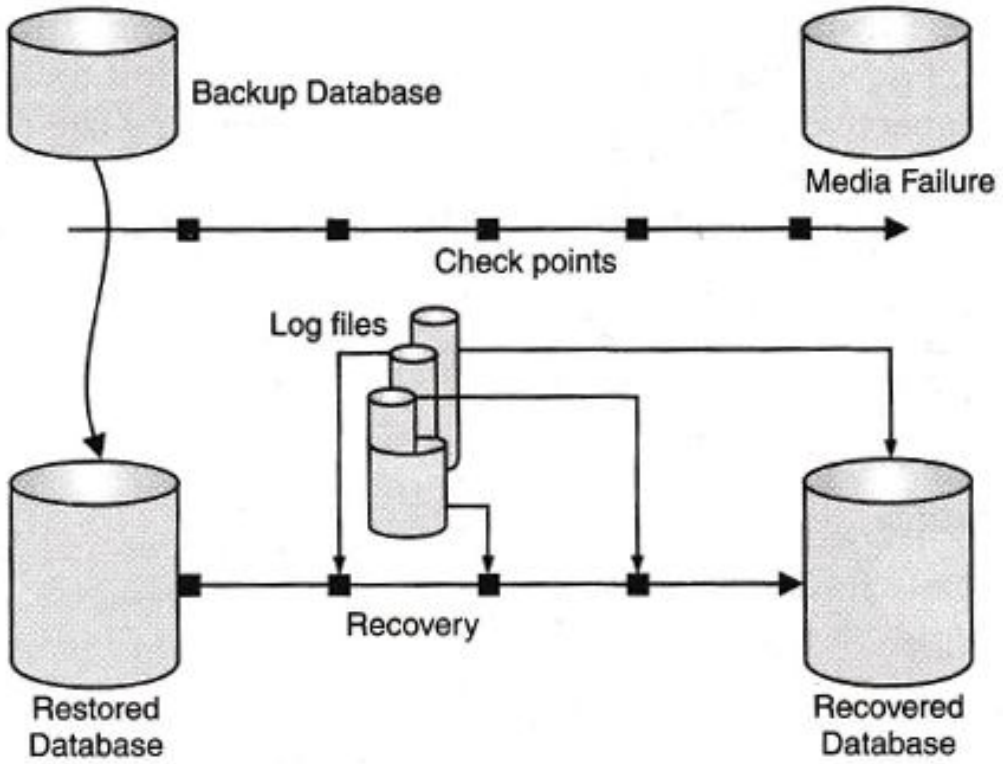
Log	Database
$<T_0$ start>	
$<T_0$ A, 1000, 950>	
$<T_0$ B, 2000, 2050>	A = 950
$<T_0$ commit>	B = 2050
$<T_1$ start>	
$<T_1$ C, 700, 600>	C = 600
$<T_0$ commit>	

This figure shows one possible order in which the actual outputs took place in both the database system and the log as a result of the execution of T_0 and T_1.

Shadow Paging:

The shadow paging scheme does not require the use of a log file in a single user environment, though a log is needed in multi-user environment for concurrency control. Shadow paging considers the database to be made up of a number of fixed-size disk pages or disk blocks for recovery purposes.

The shadow paging technique maintain two directories (two for each database page) during the life of a transaction – a current directory and a shadow directory. When the disk and the current directory is used by the transaction. The shadow directory is saved to the disk and the current directory is used by the transaction. The shadow directory is never changed and is used to restore the database in case of a failure.
During the transaction execution, the shadow directory is never modified.



Recovery with concurrent transaction:

Recovery from concurrent transaction can be done in the various four ways:

1. **Interaction with concurrency control :** In this scheme, the recovery scheme depends greatly on the concurrency control scheme that is used. So, rollback a failed transaction, we must undo the updates performed by the transaction.

2. **Transaction Rollback :** In this scheme we rollback a failed transaction by using the log. The system scans the log backward, for every log record log record found in the log the system restores the data item.
3. **Check points :** In this scheme we used check points to reduce the number of log record that the system must scan when it recovers from a crash. In a concurrent transaction-processing system, we require that the checkpoint log record be of the from <checkpoint L>, where 'L' is a list of transactions active at the time of the check point.
   A fuzzy chekpoint is a checkpoint where transactions are allowed to perform updates even while buffer blocks are being written out.
4. **Restart Recovery:** When the system recovers from a crash, it constructs two lists. The undo-list consists of transactions to be undone, and the redo-list consists of transaction to be redone. The system constructs the two lists as follows : Initially, they are both empty. The system scans the log backward, examining each record, until it finds the first <check point> record.

Example : What benefit does rigorous-two phase locking provide? How does it compare with other forms of two-phase locking?

Soln. Rigorous two-phase locking has the advantages of strict 2PL. In addition it has the property that for two conflicting transactions, their commit order is their serializability order. In some systems users might expect this behaviour.

Example : Show by example that there are schedules possible under the tree protocol that are not possible under the two-phase locking protocol, and vice-versa.

Soln. Consider the tree-structured database graph given below:

Schedule possile under three protocol but not under 2PL

T_1	T_2
lock(A)	
lock(B)	
unlock(A)	
	lock(A)
lock(C)	
unlock(B)	lock(B)
	unlock(A)
	unlock(B)
unlock(C)	

Schedule possible under 2PL but not under tree protocol:

T_1	T_2
lock(A)	
	lock(B)
lock(C)	
	unlock(B)
unlock(A)	
unlock(C)	

SOLVED PROBLEMS

1. Under the 2-phase locking protocol, which is true?
   (a) Once a transaction has released a lock it cannot acquire any more locks
   (b) A transaction can release all its locks when it commits
   (c) A transaction can acquire all the locks it needs when it begins
   (d) All these above

Soln. Regarding 2-phase locking protocol all the statements are true.
Correct option is (d)

2. Which of the following statement(s) is/are true about tree locking protocol/ graph based protocol?
   I. All the schedules that are legal under Tree locking protocol are conflict serializable.
   II. Tree locking protocol may not always ensure deadlock free
   III. Tree locking protocol always ensure recoverable and cascading schedules.
   (a) Only I is correct (b) Only II is correct
   (c) Both I & II are correct (d) All are correct

Soln. If our schedule follows all tree based locking protocol rules then our schedule is conflict serializable.
In tree based locking protocol, we allow every transaction to follow the same order while locking dada items, so there is no chance of deadlock.
In tree based locking protocol, there is chance of dirty read to happen as unlocking of data items can happen in anytime. So, it does not always ensure recoverable and cascade less schedules.
Correct option is (a)

3. Which protocol is satisfied by following transaction:

   LOCK – X(A)
   LOCK – S(B)
   R(A)
   R(B)
   W(A)
   UNLOCK(A)
   COMMIT
   UNLOCK(B)
   (a) Strict 2PL (b) Rigorous 2PL (c) Conservative 2PL (d) None of the above

Soln. The given transaction follows conservative 2PL as before stating the transaction, we obtained all the locks that we require and then we released all the locks after we used. It does not follow strict 2PL, as in strict 2PL we release all exclusive locks after committing transaction. But here 'A" which is exclusive lock is unlocked after we commit the transaction. As it does not follow strict 2PL, it does not follow rigorous 2PL also.
Correct option is (c)

Indexs File Systems

- Index File (same idea as textbook index) : auxiliary structure designed to speed up access to desired data.
- Indexing field: field on which the index file is defined.
- Index file stores each value of the index field along with pointer: pointer(s) to block(s) that contain record(s) with that field value or pointer to the record with that field value:

  <Indexing Field, Pointer>

  • In oracle, the pointer is called RowID which tells the DBMS where the row (record) is located (by file, block within that file, and row within the block).
- To find a record in the data file based on a certain selection criterion on an indexing field, we initially access the index file, which will allow the access of the record on the data file.
- Index file much smaller than the data file => searching will be fast.
- Indexing important for file systems and DBMSs:
  - Databases eventually map data to file structures on disk :
    - Records of each relation may be stored in a separate file.
    - Records of several different relations can be stored in the same file (i.e. physically clustered file organization : to minimize I/O)
  - In DBMSs, the query processor accesses the index structures for processing a query (e.g., indexed join called also single-loop join).

Ordered Indices :

- Each index structure is associated with a particular search key.
- An ordered index stores the values of the search keys in stored order and associates with each search key the records that contain it. A file may have several indices, on different search keys.
- If the file containing the records is sequentially ordered, a primary index is an index whose search key also defines the sequential order of the file.
- Primary indices are also called clustering indices.
- The search key of a primary index is usually the primary key. Indices whose search key specifies an order different from the sequential order of file are called secondary indices or non-clustering indices .

Primary Index :

- Files with a primary index on the search key, are called index-sequential files.
- Shows a sequential file of account records.
- In the example, the records are stored in search-key order, with branch-name used as the search key.