CHAPTER 11

DATABASE RECOVERY SYSTEM

After reading this chapter, the reader will understand:

- *Types of failures that can occur in a system*
- *Caching of disk pages containing data items*
- *How system log is maintained during normal execution so that the recovery process can be performed*
- *The process of checkpointing, which helps in reducing the time taken to recover from a crash*
- *The two categories of recovery techniques, namely, log-based recovery techniques and shadow paging*
- *The further classification of log-based techniques into two types, namely, techniques based on deferred update and techniques based on immediate update*
- *Recovery of a system with multiple concurrent transactions*
- *ARIES recovery algorithm*
- *Recovery of a system from catastrophic failures*

In every computer system, there is always a possibility of occurrence of a failure, which may result in loss of information. Therefore, if a failure occurs, then it is the responsibility of database management system (DBMS) to recover the database as quickly, and with as little damaging impact on users, as possible. The main aim of recovery is to restore the database to the most recent consistent state. The **recovery manager** component of DBMS is responsible for performing the recovery operations.

The recovery manager ensures that the two important properties of transactions, namely, *atomicity* and *durability* are preserved. It preserves atomicity by undoing the actions of uncommitted transactions and durability by ensuring that all the actions of committed transactions survive any type of failure. The recovery manager keeps track of all the changes that were applied to the data by various transactions and thus, deals with various states of the database. The recovery manager must also provide the **high availability** to its users, which requires the minimum levels of services and robustness in case of failures.

The recovery depends on the type of failure and the files of the database affected by the failure. This chapter first discusses the various types of failures that can cause abnormal termination of transactions, and then discusses the steps that the system takes during normal execution of transactions for recovery purpose. It then discusses how recovery is performed in case of system crashes. Finally, it discusses how the data is recovered in case of disk failures and natural disasters.

11.1 TYPES OF FAILURES

There are several types of failures that can occur in a system and stop the normal execution of the transactions. Each of them is handled in a different manner. Let us discuss these failures.

- **Logical error:** Transactions may fail due to a logical error in the transaction such as incorrect input, integer overflow, division by zero, etc. Certain exceptional conditions that are not programmed correctly may also result in cancellation of the transaction. For example, insufficient balance in a customer account results in the failure of fund withdrawal transaction. If these types of exceptions are checked within the transaction itself, they do not result in transaction failure.
- **System error:** Any undesirable state of the system such as deadlock, incorrect synchronization, etc., may also stop the normal execution of the transactions.
- **Computer failure (system crash):** Any type of hardware malfunctioning such as RAM failure, error in application software, bug in operating system, and network problem can also bring the transaction processing to a halt. It is assumed that such a failure results in the loss of data in volatile storage and does not affect the contents of the non-

volatile storage media such as disks. The assumption that the hardware and software errors bring only the system to a halt and has no effect on the contents of the non-volatile storage media is known as **fail-stop assumption**.

- **Disk failure:** Disk failure, also known as **media failure**, refers to the loss of data in some disk blocks because of disk read/write head crash, power disruption or error in data transfer operation. These errors also cause the abnormal termination of the transaction if occurred during a read/write operation of the transaction.
- **Physical problems and environment disasters:** The physical problems include theft, fire, sabotage, accidental overwriting of secondary storage media, etc. The environment disasters include floods, earthquakes, tsunami, etc.

NOTE The transaction failure and system crash are more common types of failures as compared to disk failure and natural disasters.

In case of a system crash, the information residing in the volatile storage such as main memory and cache memory is lost. Therefore, transaction failure and system crash are non-catastrophic failures as they do not result in the loss of non-volatile storage. On the other hand, disk failure and environment disasters are catastrophic failures as they result in the loss of non-volatile storage.

11.2 CACHING OF DISK PAGES

Whenever a transaction needs to update the database, the disk pages (or disk blocks) containing the data items to be modified are first cached (buffered) by the cache manager into the main memory and then modified in the memory before being written back to the disk. A **cache directory** is maintained to keep track of all the data items present in the buffers.

When an operation needs to be performed on a data item, the cache directory is first searched to determine whether the disk page containing the data item resides in the cache. If it is not present in the cache, the data item is searched on the disk and the appropriate disk page is copied in the cache. Sometimes it may be necessary to replace some of the disk pages to create space for the new pages. Any page-replacement strategy such as least recently used (LRU) or first-in-first-out (FIFO) can be used for replacing the disk page. Each memory buffer has a free bit associated with it which indicates whether the buffer is free (available for allocation) or not. Other associated bits are dirty bit and pin/unpin bit.

The data which is modified in the cache can be copied back to the disk using either of the two strategies, namely, *in-place updating* and *shadow paging*. In **in-place updating** strategy, single copy of the data items are maintained on the disk. The updated buffer is written back to the same original disk location and thus, the old value of any updated data item on the disk is overwritten by the new value. However, in **shadow paging,** multiple copies of the data items can be maintained on the disk. The updated buffer is written at a different location on the disk and thus, the old value of the data item is not overwritten by the new value.

Standard DBMS recovery terminology includes the terms steal/no-steal and force/no-force, which specify when a modified page in the cache buffer can be written back to the database on disk.

- **Steal:** In this approach, an updated cache page may be written to the disk before the transaction commits. It gives more freedom in selecting buffer pages to be replaced by the cache manager.
- **No-steal:** In this approach, an updated cache page cannot be written to the disk before the transaction commits. The pin bit is set until the updating transaction commits.
- **Force:** In this approach, all the updated cache pages are immediately written (force-written) to disk when the transaction commits.
- **No-force:** In this approach, the updated cache pages are not necessarily written to disk immediately when the transaction commits.

Things to Remember

Steal/no-force approach is the most desirable approach in practice as it gives maximum degree of freedom to the cache manager in selecting the victim pages for replacement, and in scheduling the writes to disk.

11.3 RECOVERY RELATED STEPS DURING NORMAL EXECUTION

When a DBMS is restarted after a crash, the control is given to the recovery manager, which is responsible to bring the database to a consistent state. To enable it to perform its task in the event of failure, the recovery manager maintains some information during the normal execution of transactions. It maintains a system log of all the modifications to the database and stores it on the stable storage, which is guaranteed to survive failures. The stable storage is implemented by storing the database on a number of separate non-volatile storage devices such as disks or tapes (perhaps in different locations). Information residing in stable storage is assumed to be *never* lost (in real life *never* cannot be guaranteed).

The amount of work involved during recovery depends on the changes made by the committed transactions that have not been written to the disk at the time of crash. To reduce the time to recover from a crash, the DBMS periodically force-write all the modified buffer pages during normal execution. A process called **checkpointing** also helps in reducing the time taken to recover from a crash.

11.3.1 The System Log

During the normal transaction processing, the system stores relevant information in a log to make sure that enough information is available to recover from failures. A **log** is a sequence of **log records** that contains essential data for each transaction which has updated the database. The various types of log records that are maintained in a log are listed here.

- **Start record:** The log record `[Ti, start]` is used to indicate that the transaction T_i has started.

- **Update log record:** The log record `[`$\mathtt{T_i}$`,X,`$\mathtt{V_o}$`,`$\mathtt{V_n}$`]` is used to indicate that the transaction T_i has performed an update operation on the data item `X`, having the old value $\mathtt{V_o}$ and new value $\mathtt{V_n}$.
- **Read record:** The log record `[`$\mathtt{T_i}$`,X]` is used to indicate that the transaction T_i has read the data item `X` from the database.
- **Commit record:** The log record `[`$\mathtt{T_i}$`,commit]` is used to indicate that the transaction T_i
- **Abort record:** The log record `[`$\mathtt{T_i}$`,abort]` is used to indicate that the transaction T_i has aborted, and needs to be rolled back.

Whenever a transaction needs to update a data item, two types of log entry information, namely, *UNDO-type log entry* and *REDO-type log entry* are maintained. The **UNDO-type log entry** includes the **before-image (BFIM)** of the data item which is used to undo the effect of an operation on the database. A **REDO-type log entry**, on the other hand, includes the **after-image (AFIM)** of the data item which is used to redo the effect of an operation on the database, in case the system fails before reflecting the new value of the data item to the database.

A **BFIM** is a copy of the data item prior to modification and an **AFIM** is a copy of the data item after modification.

Before making any changes to the database, it is necessary to force-write all log records to the stable storage. This is known as **write-ahead logging (WAL)**. In general, write-ahead logging protocol states that

1. A transaction is not allowed to update a data item in the database on the disk until all its UNDO-type log records have been force-written to the disk.
2. The transaction is not allowed to commit until all its REDO-type and UNDO-type log records have been force-written to the disk.

If the system fails, we can recover the consistent state of the database by examining the log and using one of the recovery techniques that are discussed in Section 11.4.

WAL is required only for in-place updating. In shadowing, since both BFIM and AFIM of the data item to be modified are kept on the disk, it is not necessary to maintain a log for recovery.

To understand how log records are maintained for a transaction, consider a transaction T_1 that transfers $100 from account A to account B. Suppose accounts A and B have initial values $2000 and $1500, respectively. The transaction T_1 is given here.

```
T1: read(A);
    A:= A – 100;
    write(A);
    read(B);
    B:= B + 100;
    write(B);
```

The log records for the transaction T_1 are shown in Figure 11.1.

For now, we assume that each time a log record for a transaction is created it is immediately written to the stable storage. However, writing each log record individually imposes an extra overhead on the system. To reduce this overhead, multiple log records are first collected in the log buffer in main memory and then copied to the stable storage in a single write operation. This is called **log record buffering**. Thus, a log record resides in the main memory for some time until it is written to the stable storage. The process of writing the buffered log to disk is known as **log force**.

```
[T1, start]
[T1,A]
[T1, A, 2000, 1900]
[T1,B]
[T1, B, 1500, 1600]
[T1, commit]
```

Fig. 11.1 *Log records for transaction T_1*

The order of log records in the stable storage must be exactly the same as that in the log buffer.

Note that during transaction rollback, only write operations are undone. The read operations are recorded in the log only to determine whether the cascading rollback is required or not. In practice, it is very complex and time-consuming to handle cascading rollback. Therefore, the recovery algorithms are designed in such a way that cascading rollback is never required. Hence, there is no need to record any read operations in the log. From now onwards, we will show the log records without the read operations.

11.3.2 Checkpointing

When the system restarts after a failure, the entire log needs to be scanned to determine the transactions that need to be redone and undone. The main disadvantage of this approach is that scanning of entire log is time-consuming. Moreover, the committed transactions that have already written their updates on the database also need to be redone as there is no way to find out the extent to which the changes of committed transactions have been propagated to the database. This causes recovery to take longer time.

To reduce the time for recovery by avoiding redo of the unnecessary work, checkpoints are used. The checkpoint approach is an additional component of the logging scheme. A **checkpoint** is periodically written into the log. It is typically written at a point when the system writes out all the modified buffers to the database on the disk. As a result, the transactions that have committed prior to the checkpoint will have their `[Ti, commit]` record before the `[checkpoint]` record. The write operations of these transactions need not be redone in case of a failure as all updates are already recorded in the database on disk. When a checkpoint is taken, the following actions are performed.

1. The execution of the currently running transactions is suspended.
2. All the log records currently residing in the main memory are written to the stable storage.
3. All the modified buffer blocks are force-written to the disk.
4. A `[checkpoint]` record is written to the log on the stable storage.
5. The execution of suspended transactions is resumed.

If the number of blocks in the buffer are large, force-writing of all these pages may take long time to finish. The time taken to force-write all modified buffers can result in undesirable delay in the processing of current transactions. For reducing this delay, a technique called fuzzy checkpointing is used in practice. In fuzzy checkpointing, a `[checkpoint]` record (known as **fuzzy checkpoint**) is written in the log before writing the modified buffer blocks to the disk. The system is then allowed to resume the execution of other transactions without having to wait for the completion of step 3. However, a system failure may occur while modified buffer blocks are being written to the disk. In that case, the fuzzy checkpoint would no longer be valid. Thus, the system is required to keep track of the previous checkpoint for which all the actions (1 to 5) have been successfully performed. For this, the system maintains a pointer to that checkpoint and it is updated to point to new checkpoint only when all the modified buffer blocks have been written to the disk.

The transactions committed before the checkpoint time need not be considered during recovery process. This reduces the amount of work required to be done during recovery.

11.4 RECOVERY TECHNIQUES

The recovery techniques are categorized mainly into two types, namely, *log-based recovery techniques* and *shadow paging*. The **log-based recovery** techniques maintain transaction logs to keep track of all update operations of the transactions. On the other hand, the **shadow paging** technique does not require the use of a log as both the after image and before image of the data item to be modified are maintained on the disk. This section discovers the recovery techniques for the situations where transactions execute serially one after the other and only one transaction is active at a time. Recovery in situations where multiple transactions execute concurrently is discussed in Section 11.5.

11.4.1 Log-based Recovery

The log-based recovery techniques are classified into two types, namely, techniques based on deferred update and techniques based on immediate update. In **deferred update** technique, the transaction is not allowed to update the database on disk until the transaction enters into the partially committed state. In **immediate update** technique, as soon as a data item is modified in cache, the disk copy is immediately updated. That is, the transaction is allowed to update the database in its active state.

To recover from a failure, basically two operations, namely, *undo* and *redo* are applied with the help of the log on the last consistent state of the database. The undo operation reverses (rolls back) the changes made to the database by an uncommitted transaction and restores the database to the consistent state that existed before the start of transaction.

Learn More

Undoing the effect of only some (not all) uncommitted transctions is known as **partial undo.** Similarly, redoing the effect of only some committed transactions is known as **partial redo**.

The redo operation reapplies the changes of a committed transaction and restores the database to the consistent state it would be at the end of the transaction. The redo operation is required when the changes of a committed transaction are not or partially reflected to the database on disk. The redo modifies the database on disk to the new values for the committed transaction.

Sometimes the undo and redo operations may also fail due to any reason, and this type of failure can occur any number of times before the recovery is completely successful. Therefore, the undo and redo operations for a given transaction are required to be **idempotent**, which implies that executing any of these operations several times must be equivalent to executing it once. That is,

undo(operation)=undo(undo(…undo(operation)…))

redo(operation)=redo(redo(…redo(operation)…))

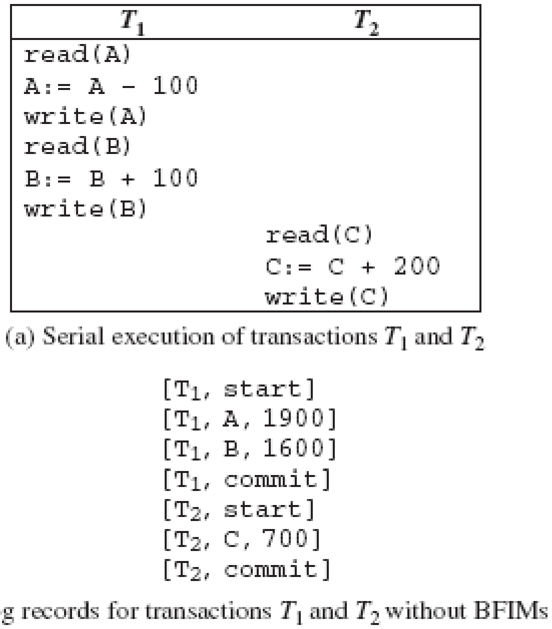
Recovery Techniques Based on Deferred Update

The deferred update technique records all update operations of the transactions in the log, but postpones the execution of all the update operations until the transactions enter into the partially committed state. Once the log records are written to the stable storage under WAL protocol, the database on the disk is updated.

If the transaction fails before reaching its commit point no undo is required as the transaction has not affected the database on the disk. In this case, the information in the log is simply ignored and the failed transaction must be restarted either automatically by the recovery process or manually by the user. However, redo operation is required in case the system fails after the transaction commits but before all the updates are reflected in the database on disk. In such situation, the log is used to redo all the transactions affected by this failure. Thus, this is known as **NO-UNDO/REDO recovery algorithm**. Note that the update log records, in this case, only contain the AFIM of the data item to be modified. The BFIM of the data item is omitted, as no undo operation is required in case of a failure. This technique uses **no-steal/no-force** approach for writing the modified buffers to the database on disk.

For example, consider the transactions T_1 and T_2 given in Figure 11.2(a). The transaction T_1 transfers $100 from account A to account B and transaction T_2 deposits $200 to account C. Assume that the initial values of account A, B, and C are $2000, $1500, and $500, respectively. Suppose that these two transactions are executed serially in the order T_1 followed by T_2. The corresponding log records (only for write operations) are shown in Figure 11.2(b).

If the system fails before writing the log record `[T₁, commit]`, no redo is required as no `commit` record is found in the log. The values of account A and B remain $2000 and $1500, respectively. If the system fails after writing the log record `[T₁, commit]`, but before writing the `commit` record for the transaction T_2, only `redo(T₁)` operation is performed. If the system fails after writing the log record `[T₂, commit]`, both T_1 and T_2 need to be redone. In general, for each commit record `[Tᵢ, commit]` found in the log, the operation `redo(Tᵢ)` is performed by the system.



T_1	T_2
read(A)	
A:= A - 100	
write(A)	
read(B)	
B:= B + 100	
write(B)	
	read(C)
	C:= C + 200
	write(C)

(a) Serial execution of transactions T_1 and T_2

```
[T1, start]
[T1, A, 1900]
[T1, B, 1600]
[T1, commit]
[T2, start]
[T2, C, 700]
[T2, commit]
```

(b) Log records for transactions T_1 and T_2 without BFIMs

Fig. 11.2 *Transactions and their log records*

Recovery Techniques Based on Immediate Update

The immediate update technique allows a transaction to update the database on the disk immediately, even before it enters into partially committed state. That is, a transaction can update the database while it is in the active state. The data modifications made by the active transactions are known as **uncommitted modifications**. Like deferred update technique, immediate update technique also records the update operations in the log (on the stable storage) first so that the database can be recovered after a failure.

If the system fails before the transaction enters into partially committed state, all the changes made to the database by this transaction must be undone. However, if immediate update ensures that all the updates made by a transaction are recorded in the database on the disk before the transaction commits, redo operation is not required. Therefore, this recovery algorithm is known as **UNDO/NO-REDO recovery algorithm**. This technique uses **steal/force** approach for writing the modified buffers to the database on the disk.

The log records in this case contain only before-image of the data item. The after-image can be omitted, since no redo is required. The undo operation is performed by replacing the current value of a data item in the database by its BFIM stored in the log. For example, consider the transaction T_1 and T_2 given in Figure 11.2(a). The corresponding log records (only for write operations) for these transactions are shown in Figure 11.3.

```
[T1, start]
[T1, A, 2000]
[T1, B, 1500]
[T1, commit]
[T2, start]
[T2, C, 500]
[T2, commit]
```

Fig. 11.3 *Log records for transaction T_1 and T_2 without AFIMs*

If the system fails before writing the log record [T_1, commit], the operation undo(T_1) is performed. If the system fails after writing the log record [T_1, commit] but before writing the commit record for the transaction T_2, the operation undo(T_2) is performed. The operation redo(T_1) is not required as it is assumed that all updates of transaction T_1 are already reflected to the database on the disk. If the system fails after writing the log record [T_2, commit], neither undo nor redo operation is required.

There is a possibility that immediate update allows a transaction to commit before all its changes are written to the database. In this case, both undo and redo operations may be required. Therefore, this algorithm is known as **UNDO/REDO recovery algorithm**. This technique uses **steal/no-force** approach for writing the modified buffers to the database on the disk. The UNDO/REDO algorithm is the most commonly used algorithm. The log records in this case contain both the BFIM and AFIM of the data item to be modified.

For example, consider the transaction T_1 and T_2 given in Figure 11.2(a). The corresponding log records (only for write operations) containing both BFIM and AFIM of the data items to be modified are shown in Figure 11.4.

If the system fails before writing the log record [T_1, commit], the operation undo(T_1) is performed. If the system fails after writing the log record [T_1, commit] but before writing the commit record for the transaction T_2, the operations redo(T_1) and undo(T_2) are performed. Here, the transaction T_1 should be redone as all updates of T_1 may not be reflected to the database on the disk. If the system fails after writing the log record [T_2, commit], the operations redo(T_1) and redo(T_2) need to be performed.

[T_1, start]

[T_1, A, 2000, 1900]

[T_1, B, 1500, 1600]

[T_1, commit]

[T_2, start]
[T_2,C,500,700]
[T_2, commit]

Fig. 11.4 *Log records for transactions T_1 and T_2 with both BFIMs and AFIMs*

11.4.2 Shadow Paging

The shadow paging technique is different from the log-based recovery techniques in a way that it does not require the use of log in an environment where only one transaction is active at a time. However, in an environment where multiple concurrent transactions are executing, the log is maintained for the concurrency control method. We will not discuss shadow paging in an environment with multiple concurrent transactions.

Shadow paging is based on making copies (known as **shadow copies**) of the database. This technique assumes that only one transaction is active at a time and thus, can be used as an alternative to log-based recovery technique in case the transactions are executed serially. This recovery technique is simple to implement but not as efficient as log-based recovery techniques.

Shadow paging considers the database to be made up of fixed-size logical units of storage called **pages**. These pages are mapped into physical blocks of storage with the help of **page table** (or **directory**). The physical blocks are of the same size as that of the logical blocks. A page table with *n* entries is constructed in which the i^{th} entry in the page table points to the i^{th} database page on the disk.

The main idea behind this technique is to maintain two page tables, namely, *current page table* and *shadow page table* during the life of a transaction. The entries in the **current page table** (or **current directory**) points to the most recent database pages on the disk. When a transaction starts, the current page table is copied into a **shadow page table** (or **shadow directory**). The shadow page table is then saved on the disk and the current page table is used by the transaction. The shadow page table is never modified during the execution of the transaction.

Initially, both the tables are identical. Whenever a write operation is to be performed on any database page, a copy of this page is created onto an unused page on the disk. The current page table is then made to point to this copy, and the update is performed on this copy. The shadow page table continues to point to the old unchanged page. The old copy of the page is never overwritten. This implies that for the pages updated by the transactions, two versions are kept. The shadow page table points to the old version and the current page table points to the new version of the page. Figure 11.5 illustrates the concept of shadow and current page table. In this figure, pages 1 and 3 have two versions. The shadow page table references the old versions and current page table references the new versions of page 1 and 3.

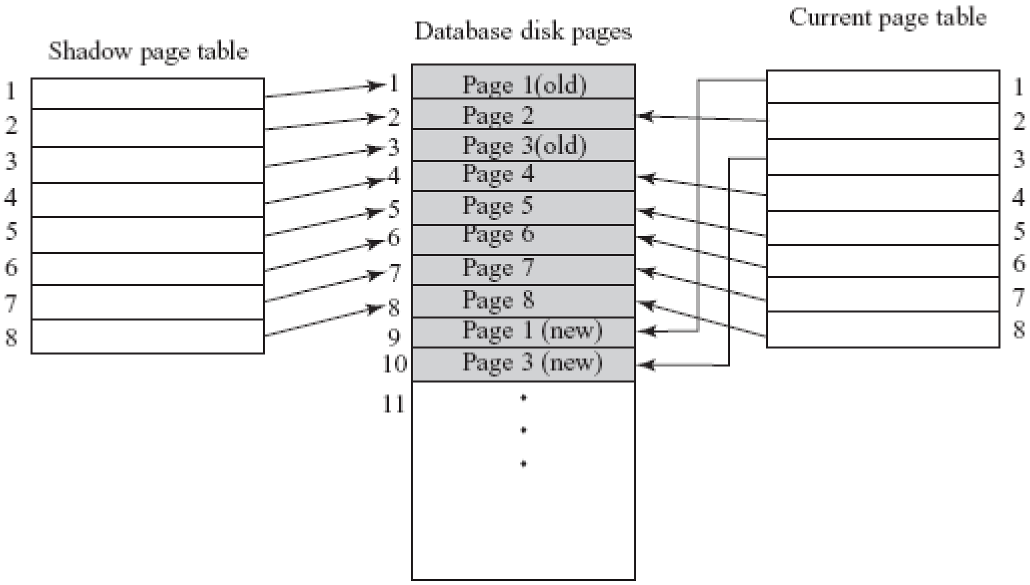


Fig. 11.5 *Shadow paging*

In case of a failure, the current page table is discarded and the disk blocks containing the modified disk pages are released. The previous state of the database prior to the transaction execution can be recovered from the shadow page table. In case no failure occurs and the transaction is committed, the shadow page table is discarded and the disk blocks with the old data are released. Since recovery in this case requires neither undo nor redo operations, this algorithm is known as **NO-UNDO/NO-REDO recovery algorithm**. This technique uses **no-steal/force** approach for writing the modified buffers to the database on the disk.

The shadow paging technique has several disadvantages. Some of them are discussed here.

- **Data fragmentation:** Shadow paging technique causes the updated database pages to change locations on the disk. This makes it difficult to keep the related database pages together on the disk.
- **Garbage collection:** Whenever a transaction commits, the database pages containing the old version of data are considered as garbage as they do not contain any usable information. Thus, it is necessary to find all of the garbage pages periodically and add them to the list of free pages. This process of finding garbage pages periodically is known as **garbage collection**. Though this process incurs an extra overhead on the system, but is necessary to improve the system performance.
- **Harder to extend:** It is difficult to extend the algorithm to allow transactions to run concurrently.

11.5 RECOVERY FOR CONCURRENT TRANSACTIONS

The recovery techniques discussed so far handle the transactions in an environment where transactions execute serially. This section discusses how the log-based recovery algorithms can be extended to handle multiple transactions running concurrently. Note that regardless of the number of transactions, the system maintains a single log for all the transactions.

In case, where several transactions execute concurrently, the recovery algorithm may be more complex, depending on the concurrency control technique being used. In general, the greater the degree of concurrency, the more time-consuming the task of recovery becomes.

Consider a system in which strict two-phase locking protocol is used for concurrency control. To minimize the work of a recovery manager, checkpoints are also maintained at regular intervals. Two lists, namely, *active list* and *commit list* are maintained by the recovery system. All the active transactions T_A are entered in the **active list** and all the committed transactions T_C since the last checkpoint are entered in the **commit list**.

First, consider the case when UNDO/REDO recovery algorithm is used. During recovery process, the write operations of all the transactions in the commit list are redone in the order in which they were written to the log. The write operations of all the transactions in the active list are undone in the reverse of the order in which they were written to the log.

For example, consider five transactions T_1, T_2, T_3, T_4, and T_5 executing concurrently as shown in Figure 11.6. Suppose a checkpoint is made at time t_c and the system crash occurs at time t_f. During the recovery process, transactions T_2 and T_3 (present in commit list) need to be redone, as they are committed after the last checkpoint. Since the transaction T_1 is committed before the last checkpoint, its operations need not be redone. The transactions T_4 and T_5 (present in the active list) were not committed at the time of system crash and hence, need to be undone.

If NO-UNDO/REDO algorithm is followed, the transactions in the active list are simply ignored, as these transactions have not affected the database on the disk and hence, need not be undone. Similarly, if UNDO/NO-REDO technique is followed, the transactions in the commit list are ignored as their effects are already reflected in the database on disk.

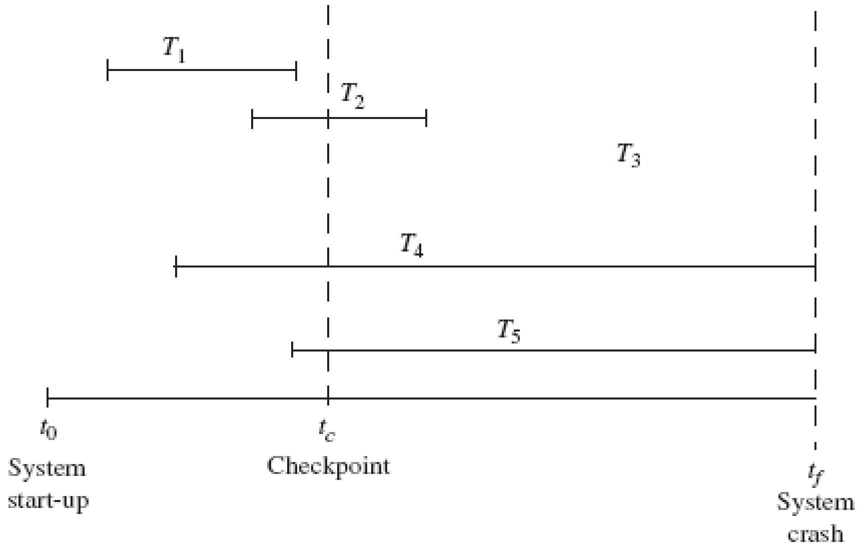


Fig. 11.6 *Recovery with concurrent transactions*

In case a data item `Q` is modified by different transactions, all the updates made to `Q` are overwritten by the last `write` operation. The NO-UNDO/REDO algorithm can be made more efficient by only redoing the last update of `Q`, rather than redoing all the update operations. In this situation, the log is scanned in the backward direction, starting from the end. As soon as a `write` operation on a data item `Q` is redone, `Q` is added in a list of redone items. Now, before applying a redo operation on any data item, it is first searched in the list of redone items. If the item is found in the list, it is not redone again as its last value has already been recovered.

11.6 ARIES RECOVERY ALGORITHM

Algorithm for Recovery and Isolation Exploiting Semantics (ARIES) is an example of recovery algorithm which is widely used in the database systems. It uses steal/no-force approach for writing the modified buffers back to the database on the disk. This implies that ARIES follows UNDO/REDO technique. The ARIES recovery algorithm is based on three main principles which are given here.

- **Write-ahead logging:** This principle states that before making any changes to the database, it is necessary to force-write the log records to the stable storage.

- **Repeating history during redo:** When the system restarts after a crash, ARIES retraces all the actions of database system prior to the crash to bring the database to the state which existed at the time of the crash. It then undoes the actions of all the transactions that were not committed at the time of the crash.
- **Logging changes during undo:** A separate log is maintained while undoing a transaction to make sure that the undo operation once completed is not repeated in case the failure occurs during the recovery itself, which causes restart of the recovery process.

In ARIES, each log record is assigned a unique id called the **log sequence number (LSN)** in monotonically increasing order. This id indicates the address of the log record on the disk. The ARIES algorithm splits a log into a number of sequential log files, each of which is assigned a file number. When a particular log file grows beyond a certain limit, a new log file is created and new records are added to that file. The new file is assigned a number higher by 1 than the previous log file. In this case, the LSN consists of a file number and an offset within the file. Every data page has a **pageLSN** field that is set to the LSN of the log record corresponding to the last update on the page. During the redo operation, the log records with LSN less than or equal to pageLSN of the page should not be executed as their actions are already reflected in the database.

The set of all log records for a particular transaction are stored in the form of linked list. Every log record includes some common fields such as previous LSN (prevLSN), transaction ID, and type of the log record. The **prevLSN** contains the address of the previous log record of the transaction. It is required to traverse the linked list in backward direction. The **transaction ID** field contains the id of the transaction for which the log record is maintained. The **type** field contains the type of the log record. The update log record also contains some additional fields such as **pageID** containing the data item, **length** of the updated item, its **offset** from the beginning of the page, and the **BFIM** and **AFIM** of the data item being modified.

In addition to the log, two tables, namely, *transaction table* and *dirty page table* are also maintained for efficient recovery. The **transaction table** contains a record for each active transaction. The record contains the information such as transaction ID, transaction status (active, committed, or aborted), and the LSN of the most recent log record (called **lastLSN**) for the transaction. The **dirty page table** contains a record for each dirty page in the buffer. Each record consists of a pageID and recLSN. The **recLSN** is the LSN of the first log record that has updated the page. This LSN gives the earliest log record that might have to be redone for this page when the system restarts after a crash. During normal operation, the transaction table and the dirty page table are maintained by the transaction manager and buffer manager, respectively.

The most recent portion of the log, which is kept in main memory, is called the **log tail**. The log tail is periodically forced-written to the stable storage. A log record is written for each of these actions.

- updating a page (write operation)
- committing a transaction
- aborting a transaction
- undoing an update
- ending a transaction

The log records for update operation, and committing and aborting a transaction have already been discussed. For undoing an update operation, a **compensation log record (CLR)** is maintained that records the actions taken during the rollback of a particular update operation. It is appended to the log tail as any other log record. The CLR contains an additional field, called the **UndoNextLSN**, which contains the LSN of the log record that needs to be undone next, when the transaction is being rolled back. Unlike update log records, the CLRs describe the actions that have already been undone and hence, will not be undone again because we never need to undo an action that has already been undone. The number of CLRs written during undo is same as of the number of update log records for transactions that were not committed at the time of crash.

When a transaction is committed or aborted, some additional actions (like removing its entry from the transaction table, etc.) are taken after writing the commit or abort log record. When these actions are completed, the **end** (or **completion**) **log record** containing the transaction id is appended in the log. Thus, this record notes that a particular transaction has been committed or aborted.

A checkpoint is also maintained by writing a `begin_checkpoint` and `end_checkpoint` record in the log. The LSN of the `begin_checkpoint` record is written to a special file which is accessed during recovery to find the last checkpoint information. When an `end_checkpoint` record is written, the contents of both the transaction table and the dirty page table are appended to the end of the log. To reduce the cost of checkpointing, fuzzy checkpointing is used so that DBMS can continue to execute transactions during checkpointing.

The contents of the modified cache buffers need not be flushed to disk during checkpoint. This is because the transaction table and dirty page table, which are already appended to log on the stable storage, contain the information needed for recovery. Note that if the system fails during checkpointing, the special file containing the LSN of `begin_checkpoint` record of the previous checkpoint will be used for recovery.

When a system restarts after a crash, the ARIES recovers from the crash in three phases:

- **Analysis phase:** This phase starts from the `begin_checkpoint` record and proceeds till the end of the log. When the `end_checkpoint` record in the log is encountered, the transaction table and dirty page table are accessed which were written in the log during checkpointing. During analysis, these two tables are reconstructed as follows:
    - If an end log record for a transaction $T i$ is encountered in the transaction table, its entry is deleted from that table.

  - If any other type of log record for a transaction $_{T}j$ is encountered, its entry is inserted into the transaction table (if not present) and the last LSN is modified accordingly.
  - If the log record specifying a change in page P is encountered, an entry is made for that page (if not already present) in the dirty page table and associated recLSN field is modified.

At the end of analysis phase, the necessary information for redo and undo phase has been compiled in transaction table and dirty page table.

- **Redo phase:** The redo phase actually reapplies the updates from the log to the database. In ARIES, the redo operation is not applied to only the committed transactions, rather, a starting point from which the redo phase should start is determined first, and from this point the redo phase begins. The smallest LSN, which indicates the log position from where the redo phase needs to be started, is determined from the dirty page table. Before this point, the changes to dirty pages have already been reflected to the database on disk.
- **Undo phase:** This phase rolls back all transactions that were not committed at the time of failure. A backward scan of the log is performed and all the transactions in the undo-list are rolled back. A compensating log record for each undo action is written in log. After undo phase, the recovery process is finished and normal processing can be started again.

Learn More

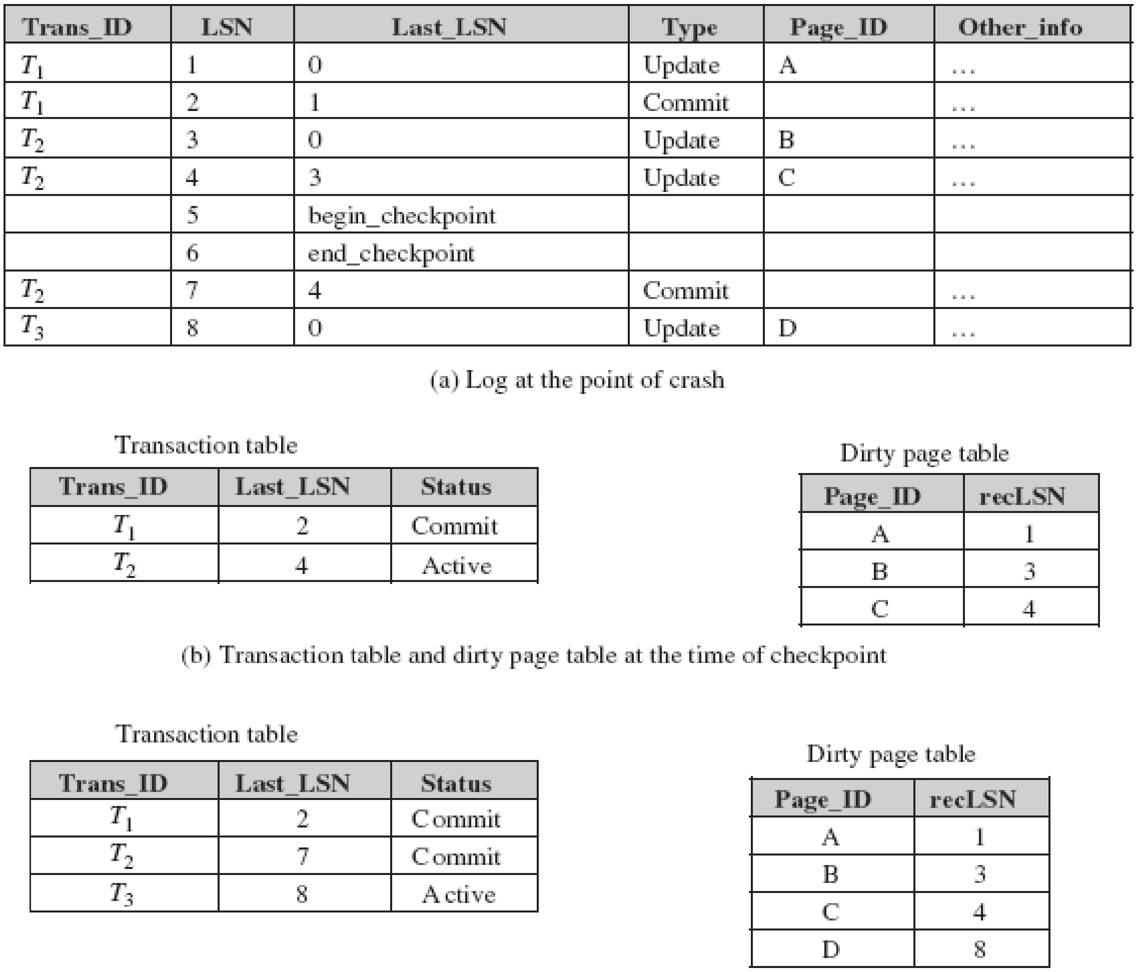
IBM DB2, MS SQL Server, and Oracle 8 use Write-ahead logging scheme for recovery from a system crash. Out of these, IBM DB2 uses ARIES, while other use schemes that are more or less similar to ARIES.

To understand how ARIES works, consider three transactions T_1, T_2, and T_3, where T_1 is updating page A, T_2 is updating pages B and C, and T_3 is updating page D. The log records at the point of crash are shown in Figure 11.7(a). The transaction table and dirty page table at the time of checkpoint and after the analysis phase are shown in Figures 11.7(b) and 11.7(c), respectively. The analysis phase starts from the `begin_checkpoint` record, which is at LSN 5 and continues until it reaches the end of the log. The transaction table and dirty page table are reconstructed during analysis phase as given below:

- When the log record 7 is encountered, the status of transaction T_2 is changed to 'committed' in the transaction table.
- When the log record 8 is encountered, new entry for the transaction T_3 is made in transaction table and an entry for page *D* is made in dirty page table.

During redo phase, the smallest LSN is determined from the dirty page table, which is 1. Thus, redo phase will start from the log record 1. The LSNs 1, 3, 4, and 8 indicate the change in page *A*, *B*, *C*, and *D*, respectively. Since these LSNs are not less than the smallest LSN, the updates on these pages need to be reapplied from the log.

During undo phase, the transaction table is scanned to determine the transactions that need to be rolled back. The transaction T_3 is the only transaction that was active at the point of crash. Thus, T_3 needs to be rolled back. The undo phase begins at log record 8 (for transaction T_3) and proceeds backward to undo all the updates performed by transaction T_3. Since in our example, log record 8 is the only update operation performed by T_3, it needs to be undone.



Trans_ID	LSN	Last_LSN	Type	Page_ID	Other_info
T_1	1	0	Update	A	…
T_1	2	1	Commit		…
T_2	3	0	Update	B	…
T_2	4	3	Update	C	…
	5	begin_checkpoint			
	6	end_checkpoint			
T_2	7	4	Commit		…
T_3	8	0	Update	D	…

(a) Log at the point of crash

Transaction table

Trans_ID	Last_LSN	Status
T_1	2	Commit
T_2	4	Active

Dirty page table

Page_ID	recLSN
A	1
B	3
C	4

(b) Transaction table and dirty page table at the time of checkpoint

Transaction table

Trans_ID	Last_LSN	Status
T_1	2	Commit
T_2	7	Commit
T_3	8	Active

Dirty page table

Page_ID	recLSN
A	1
B	3
C	4
D	8

(c) Transaction table and dirty page table after the analysis phase

Fig. 11.7 *An example of ARIES recovery algorithm*

ARIES has many advantages which are given here.

- It is simple and flexible to implement.
- It can support concurrency control techniques that involve locks of finer granularity. It also provides a variety of optimization techniques to improve concurrency control.
- It uses a number of techniques to reduce logging overhead, overheads of checkpoints, and time taken for recovery.

11.7 RECOVERY FROM CATASTROPHIC FAILURES

The recovery techniques discussed so far are applied in case of non-catastrophic failures. The recovery manager should also be able to deal with catastrophic failures. Though, such failures are very rare, still some techniques must be developed to recover from such failures. The main technique used to handle such failures is **database backup** (also known as **dump**). In this technique, the entire contents of the database and the log are copied periodically onto a cheap storage medium such as magnetic tapes.

When a database backup is taken, the following actions are performed.

1. The execution of the currently running transactions is suspended.
2. All the log records currently residing in the main memory are written to the stable storage.
3. All the modified buffer blocks are force-written to the disk.
4. The contents of the database are copied to the stable storage.
5. A `[dump]` record is written to the log on the stable storage.
6. The execution of suspended transactions is resumed.

Note that all the steps except step 4 are similar to the steps used for checkpoints. Thus, the database backup and checkpointing are similar processes to an extent. Like fuzzy checkpoints, the technique called **fuzzy dump** is also used in practice. In fuzzy dump technique, the transactions are allowed to continue while the dump is in progress.

Since the system log is much smaller than the database itself, the backup of the log records is taken more frequently than the full database backup. The backup of the system log helps in redoing the effect of all the transactions that have been executed since the last backup. In case of disk failure the most recent dump of the database is loaded from the magnetic tapes to the disk. The system log is then used to bring the database to the most recent consistent state by reapplying all the operations since the last backup.

In case of environmental disasters such as flood, earthquakes, etc., the backup of the database taken on the stable storage at a remote site is used to recover the lost data. The copy of the database is made generally by writing each block of stable storage over a computer network. This is called **remote backup**. The site at which the transaction processing is performed is known as **primary site** and the remote site at which the backup is taken is known as **secondary site**. The remote site must be physically separated from the primary site so that any kind of natural disaster does not damage the remote site.

In case of natural disasters when the primary site fails, the remote site takes over the processing. But before that the recovery is performed at the remote site using the most recent backup of the data (may be outdated) and the log records received from the primary site. Once the recovery has been performed, the remote site can start the processing of the transactions.

Note that each time the updates are performed at the primary site; the updates must be propagated to the remote site so that the remote site remains synchronized with the primary site. This can be achieved by sending all the log records from the primary site to the remote site where the backup is taken.

SUMMARY

1. The main aim of recovery is to restore the database to the most recent consistent state which existed prior to the failure. The recovery manager component of DBMS is responsible for performing the recovery operations.
2. The recovery manager ensures that the two important properties of transactions, namely, atomicity and durability are preserved.
3. There are several types of failures that can occur in a system and stop the normal execution of the transaction such as logical error, system error, computer failure (system crash), disk failure, physical problems, and environmental disasters.
4. Whenever a transaction needs to update the database, the disk pages containing the data items to be modified are first cached (buffered) by

the cache manager into the main memory and then modified in the memory before being written back to the disk.

5. A cache directory is maintained to keep track of all the data items present in the buffers.
6. The data which is modified in the cache can be copied back to the disk using either of the two strategies, namely, in-place updating and shadow paging.
7. Standard DBMS recovery terminology includes the terms steal/no-steal and force/no-force, which specify when a modified page in the cache buffer can be written back to the database disk.
8. The recovery manager maintains a system log of all modifications to the database and stores it on the stable storage.
9. A log is a sequence of log records that contains essential data for each transaction which has updated the database.
10. In case of update operation, the log record contains the before and after images of the portion of the database which have been modified by the transaction.
11. Before making any changes to the database, it is necessary to force-write all log records on the stable storage. This is known as write-ahead logging (WAL).
12. The process in which multiple log records are first collected in the log buffer and then copied to the stable storage in a single write operation is called log record buffering. Writing the buffered log to disk is known as log force.
13. To recover from a failure, basically two operations, namely, undo and redo are applied with the help of the log on the last consistent state of the database.
14. The undo operation undoes (reverses or rollbacks) the changes made to the database by an uncommitted transaction and restores the database to the consistent state that existed before the start of transaction.
15. The redo operation redoes the changes of a committed transaction and restores the database to the consistent state it would be at the end of the transaction. The redo operation is required when the changes of a committed transaction are not or partially reflected to the database on disk.

16. The recovery techniques are categorized mainly into two types, namely log-based recovery techniques and shadow paging.
17. The log-based recovery techniques maintain transaction logs to keep track of transaction operations. The log-based recovery techniques are further classified into two types, namely, techniques based on deferred update and techniques based on immediate update.
18. In deferred update technique, the transaction is not allowed to update the database on disk until the transaction enters into the partially committed state.
19. In immediate update technique, as soon as a data item is modified in cache, the disk copy is immediately updated.
20. To reduce the time to recover from a crash, the DBMS periodically force-write all the modified buffer pages during normal execution using a process known as checkpointing.
21. The shadow paging technique does not require the use of a log as both the after image and the before image of the data item to be modified are maintained on the disk.
22. In case of recovery when concurrent transactions are executing, two lists, namely, active list and commit list are maintained by the recovery system. All the active transactions are entered in the active list and all the committed transactions since the last checkpoint are entered in the commit list.
23. Algorithm for Recovery and Isolation Exploiting Semantics (ARIES) is an example of recovery algorithm which is widely used in the database systems. It uses steal/no-force approach for writing the modified buffers back to the database on the disk.
24. In ARIES, each log record is assigned a unique id called the log sequence number (LSN) in monotonically increasing order. This id indicates the address of the log record on the disk.
25. ARIES is based on three principles including write-ahead logging, repeating history during redo, and logging changes during undo.
26. When a system restarts after a crash, the ARIES recovers from the crash in three phases—the first phase is the analysis phase, second phase is the redo phase, and the last phase is the undo phase.

27. To recover from catastrophic failures which result in loss of data in non-volatile storage, the backup of the database known as dump is used. The most recent dump of the database is loaded from the magnetic tapes to the disk. The system log is then used to bring the database to the most recent consistent state by reapplying all the operations since the last backup.
28. Remote backup systems provide high degree of availability, which allows transaction processing to continue even if the primary site is destroyed by any natural disaster such as fire, flood, or earthquake.

KEY TERMS

- Recovery manager
- Logical error
- System error
- System crash
- Fail-stop assumption
- Disk failure
- Cache directory
- In-place updating
- Shadow paging
- Steal
- No-steal
- Force
- No-force
- Log
- Log records
- Start record
- Update log record
- Read record
- Commit record
- Abort record
- Undo-type log entry
- Before-image (BFIM)

- Redo-type log entry
- After-image (AFIM)
- Write-ahead logging (WAL)
- Log record buffering
- Log force
- Checkpoint
- Log-based recovery
- Deferred update
- Immediate update
- No-undo/redo recovery algorithm
- Uncommitted modifications
- Undo/no-redo recovery algorithm
- Undo/redo recovery algorithm
- Shadow copy
- Pages
- Page table
- Current page table
- Shadow page table
- No-undo/no-redo recovery algorithm
- Data fragmentation
- Garbage collection
- Active list
- Commit list
- ARIES recovery algorithm
- Log sequence number (LSN)
- Transaction table
- Dirty page table
- Compensation log record (CLR)
- End (or completion) log record
- Analysis phase
- Redo phase
- Undo phase
- Database backup (dump)
- Fuzzy dump
- Remote backup

- Primary site
- Secondary site

EXERCISES

A. Multiple Choice Questions

1. Which of these is a type of environmental disaster?
    1. Computer failure
    2. Power failure
    3. Logical error
    4. Disk failure
2. Which of these strategies can be used for copying modified data in cache to the disk?
    1. In-place updating
    2. Out-place updating
    3. Page caching
    4. Both (a) and (b)
3. Which of these terms is not included in DBMS recovery terminology?
    1. Steal
    2. Steal-force
    3. Force
    4. No-force
4. Which of these is a type of log record maintained in a log?
    1. Start record
    2. Update log record
    3. Abort record
    4. All of these
5. The UNDO/REDO recovery algorithm uses _________ approach for writing the modified buffers to the database on the disk.
    1. Steal/force
    2. No-steal/force
    3. No-steal/no-force
    4. Steal/no-force
6. Which of these terms is associated with shadow paging?

   1. Shadow copies
   2. Page directory
   3. Current directory
   4. All of these
7. The ARIES algorithm is not based on the principle of ________________.
   1. Write-ahead logging
   2. Logging changes during undo
   3. Deferred update
   4. Repeating history during redo
8. Which of these is not a phase of ARIES algorithm?
   1. Analysis phase
   2. Logging phase
   3. Redo phase
   4. Undo phase
9. For which of the following actions a log record is written?
   1. Committing a transaction
   2. Aborting a transaction
   3. Undoing an update
   4. All of these
10. Which of these actions is not performed while taking database backup?
    1. All the modified buffer blocks are force-written to the disk
    2. The contents of the database are copied to the stable storage
    3. Continue with the execution of currently running transactions
    4. A `[dump]` record is written to the log on the stable storage

B. Fill in the Blanks

1. The assumption that the hardware and software errors bring only the system to a halt and has no effect on the contents of the non-volatile storage media is known as ___________.
2. A log is a sequence of __________ that contains essential data for each transaction which has updated the database.
3. The UNDO-type log entry includes the ____________ of the data item which is used to undo the effect of an operation on the database.

4. The ____________ techniques maintain transaction logs to keep track of all update operations of the transactions.
5. In ___________ technique, as soon as a data item is modified in cache, the disk copy is immediately updated.
6. In shadow paging, the entries in the ____________ points to the most recent database pages on the disk.
7. In case of recovery when concurrent transactions are executing, the recovery manager maintains an __________ for all the active transactions and ___________ for all the committed transactions since the last checkpoint.
8. In ARIES, each log record is assigned a unique id called the ____________ in monotonically increasing order.
9. The __________ phase of ARIES algorithm actually reapplies the updates from the log to the database.
10. The site at which the transaction processing is performed is known as __________ and the remote site at which the backup is taken is known as _____________.

C. Answer the Questions

1. What is recovery manager? How does it preserve the atomicity and durability of transactions?
2. What actions does a recovery manager perform during normal execution?
3. Discuss the various types of failures that can occur in a system. Differentiate between a system crash and disk crash.
4. Define the following terms:
    1. Cache directory
    2. Page table
    3. BFIM and AFIM
    4. Active list and commit list
    5. Uncommitted modifications
5. What is the difference between in-place updating and shadow paging?
6. Differentiate between steal/no-steal and force/no-force approaches.

7. What is a system log? What are its contents? How can a log be used for database recovery? Explain with the help of an example.
8. What is write-ahead logging protocol? Discuss the two types of log entry information that are maintained during an update operation of a transaction.
9. What is log record buffering? Why is it used?
10. What are checkpoints? How does fuzzy checkpointing affect the system performance? Explain with the help of an example.
11. If the system fails, can we recover the consistent state of the database? If yes, how? Explain.
12. What are undo and redo operations? Discuss the recovery techniques that use each of the operations. Why do these operations need to be idempotent?
13. What is the difference between deferred update and immediate update techniques?
14. Discuss the recovery techniques based on deferred update. Why is it called NO-UNDO/REDO method?
15. Briefly outline the UNDO/NO-REDO algorithm in an environment where transactions execute serially one after the other.
16. Discuss the UNDO/REDO recovery algorithm in both environments where transactions execute serially and where transactions execute concurrently.
17. Discuss shadow paging. How is it different from log-based recovery techniques? Mention some of its disadvantages.
18. What is ARIES? On what principles it is based? Discuss its three phases. What do you understand by LSN in ARIES?
19. What information do the transaction table and dirty page table in ARIES contain? How are they used in efficient recovery?
20. How is the database recovered from disk failures and natural disasters?
21. Consider the following log records for the transactions T_1, T_2, T_3, T_4, and T_5. Describe the recovery process from a system crash if immediate update technique is followed. Also assume that the system maintains the checkpoints. Determine the list of transactions that need to be undone and that need to be redone.

[T_1, start]

[T_1, P, 20, 25]

[T_1, Q, 15, 35]

[T_1, commit]

[T_2, start]

[T_2, R, 50, 70]

[T_2, P, 25, 16]

[T_2, Q, 35,18]

[T_2, commit]

[checkpoint]

[T_3, start]

[T_3, R, 70, 45]

[T_4, start]

[T_4, P, 16, 22]

[T_4, Q, 18, 25]

[T_5, start]

[T_5, Q, 25, 32]

[T_5, R, 45, 25]

[T_5, commit]

[T_3, P, 22, 24]

System Crash

22. Consider the following log records containing only BFIMs of the data item being modified. Describe the recovery process from a system crash if deferred update technique is followed.

[T_1, start]

[T_1, P, 25]

[T_1, Q, 35]

[T_1, commit]

[T_2, start]

[T_2, R, 70]

[T_2, P, 16]

[T_2, Q, 18]

[T_2, commit]

[checkpoint]

[T_3, start]

[T_3, R, 45]

[T_4, start]

[T_4, P, 22]

[T_4, Q, 25]

[T_5, start]

[T_5, Q, 32]

[T_5, R, 25]

[T_5, commit]

[T_3, P, 24]

System Crash