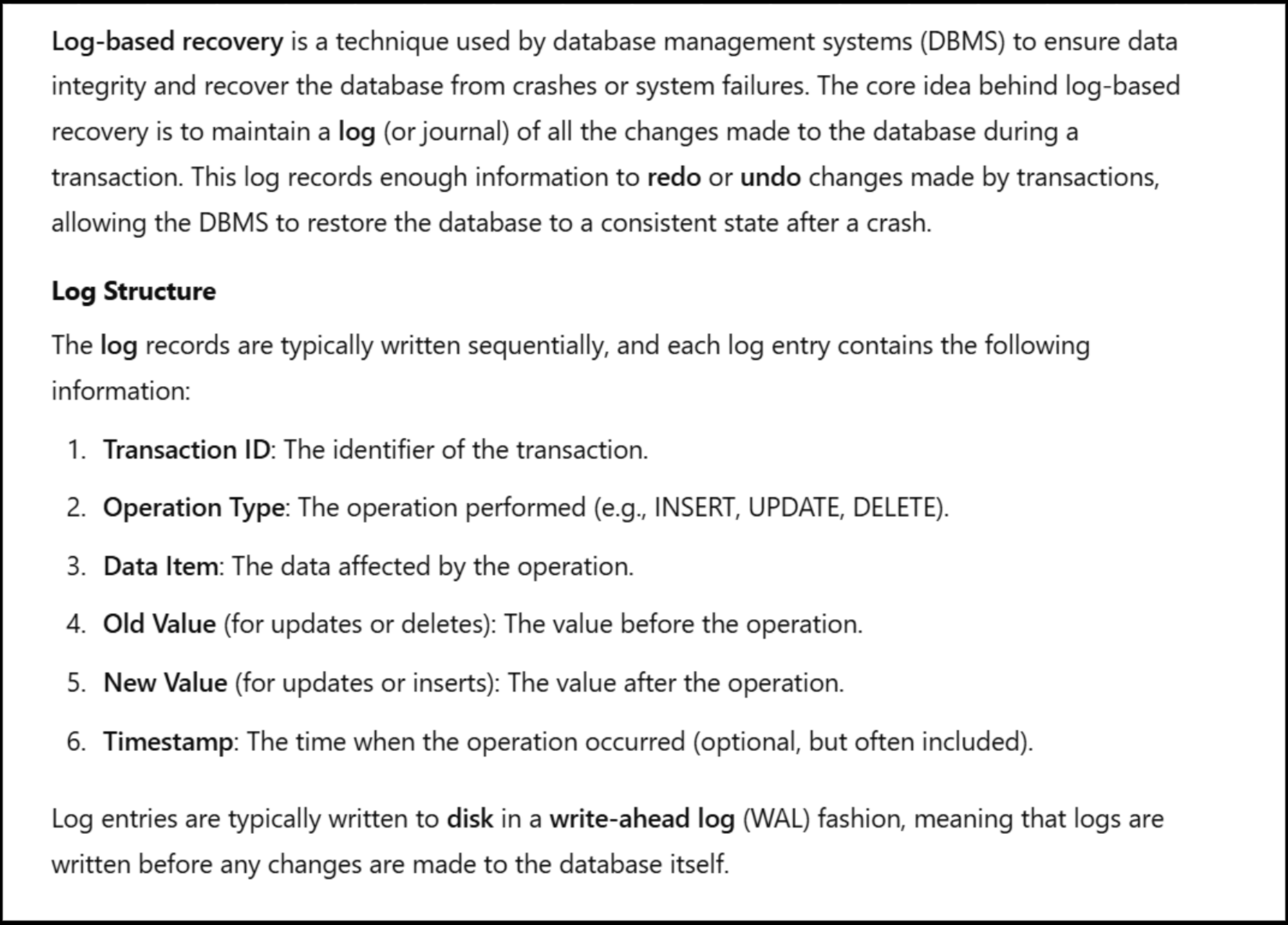
[Recovery System]

A computer system, like any other device, is subject to failure from a variety of causes: disk crash, power outage, software error, a fire in the machine room, even sabotage. In any failure, information may be lost. Therefore, the database system must take actions in advance to ensure that the atomicity and durability properties of transactions, introduced in Chapter 14, are preserved. An integral part of a database system is a **recovery scheme** that can restore the database to the consistent state that existed before the failure. The recovery scheme must also provide **high availability**; that is, it must minimize the time for which the database is not usable after a failure.

- **Transaction failure**. There are two types of errors that may cause a transaction to fail:
  - **Logical error**. The transaction can no longer continue with its normal execution because of some internal condition, such as bad input, data not found, overflow, or resource limit exceeded.
  - **System error**. The system has entered an undesirable state (for example, deadlock), as a result of which a transaction cannot continue with its normal execution. The transaction, however, can be reexecuted at a later time.
- **System crash**. There is a hardware malfunction, or a bug in the database software or the operating system, that causes the loss of the content of volatile storage, and brings transaction processing to a halt. The content of nonvolatile storage remains intact, and is not corrupted.

The assumption that hardware errors and bugs in the software bring the system to a halt, but do not corrupt the nonvolatile storage contents, is known as the **fail-stop assumption**. Well-designed systems have numerous internal checks, at the hardware and the software level, that bring the system to a halt when there is an error. Hence, the fail-stop assumption is a reasonable one.

- **Disk failure**. A disk block loses its content as a result of either a head crash or failure during a data-transfer operation. Copies of the data on other disks, or archival backups on tertiary media, such as DVD or tapes, are used to recover from the failure.



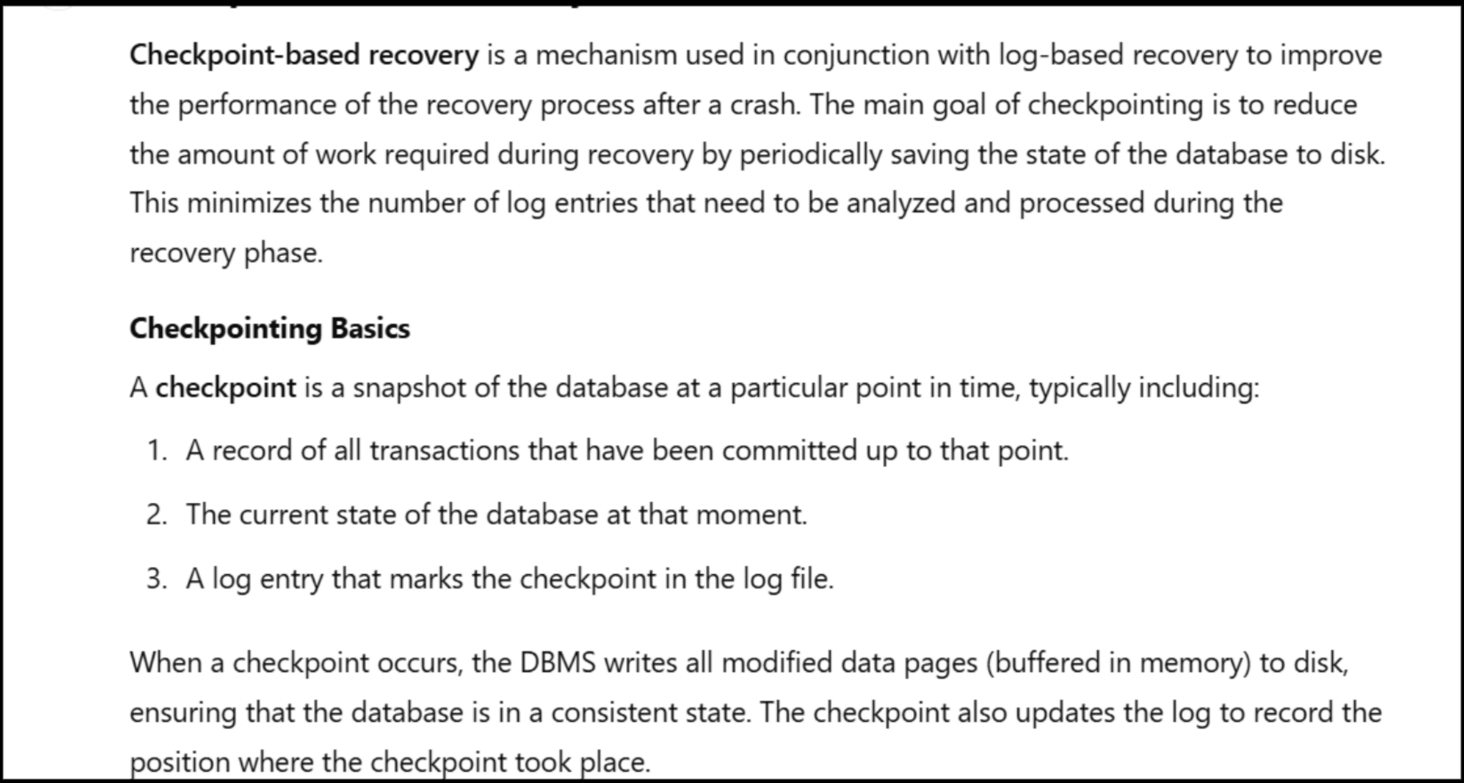
Log-based recovery is a technique used by database management systems (DBMS) to ensure data integrity and recover the database from crashes or system failures. The core idea behind log-based recovery is to maintain a **log** (or journal) of all the changes made to the database during a transaction. This log records enough information to **redo** or **undo** changes made by transactions, allowing the DBMS to restore the database to a consistent state after a crash.

Log Structure

The **log** records are typically written sequentially, and each log entry contains the following information:

1. **Transaction ID**: The identifier of the transaction.
2. **Operation Type**: The operation performed (e.g., INSERT, UPDATE, DELETE).
3. **Data Item**: The data affected by the operation.
4. **Old Value** (for updates or deletes): The value before the operation.
5. **New Value** (for updates or inserts): The value after the operation.
6. **Timestamp**: The time when the operation occurred (optional, but often included).

Log entries are typically written to **disk** in a **write-ahead log** (WAL) fashion, meaning that logs are written before any changes are made to the database itself.



Checkpoint-based recovery is a mechanism used in conjunction with log-based recovery to improve the performance of the recovery process after a crash. The main goal of checkpointing is to reduce the amount of work required during recovery by periodically saving the state of the database to disk. This minimizes the number of log entries that need to be analyzed and processed during the recovery phase.

Checkpointing Basics

A **checkpoint** is a snapshot of the database at a particular point in time, typically including:

1. A record of all transactions that have been committed up to that point.
2. The current state of the database at that moment.
3. A log entry that marks the checkpoint in the log file.

When a checkpoint occurs, the DBMS writes all modified data pages (buffered in memory) to disk, ensuring that the database is in a consistent state. The checkpoint also updates the log to record the position where the checkpoint took place.