Unit-06

Transaction Management

Transaction Concept

A transaction is a sequence of one or more operations (queries, updates, etc.) that are executed as a single unit of work. The concept of transactions is essential for maintaining the consistency, integrity, and reliability of the database. Transactions ensure that a series of operations are either completed in their entirety or leave the database in a well-defined state if an error or failure occurs.

ACID Properties

The properties of transactions are often referred to as ACID properties, which stands for Atomicity, Consistency, Isolation, and Durability:

1. Atomicity:

→ Atomicity ensures that a transaction is treated as a single, indivisible unit of work. Either all the operations within the transaction are successfully completed, or none of them are applied. If any part of the transaction fails, the entire transaction is rolled back, and the database remains unchanged.

→ For example, in an application that transfers funds from one account to another, the atomicity property ensures that, if a debit is made successfully from one account, the corresponding credit is made to the other account.

2. Consistency:

→ Consistency ensures that a transaction brings the database from one consistent state to another. The execution of a transaction should not violate any integrity constraints defined on the database. If a transaction is completed successfully, the database is left in a valid state.

→ For example, in an application that transfers funds from one account to another, the consistency property ensures that the total value of funds in both the accounts is the same at the start and end of each transaction.

3. Isolation:

→ Isolation ensures that the intermediate states of a transaction are not visible to other transactions until the transaction is committed. This property prevents interference between concurrent transactions. Each transaction should be executed in isolation, as if it were the only transaction in the system.

→ For example, in an application that transfers funds from one account to another, the isolation property ensures that another transaction sees the transferred funds in one account or the other, but not in both, nor in either.

4. Durability:

→ Durability guarantees that once a transaction is committed, its effects are permanent and survive subsequent failures. The changes made by a committed transaction are stored in a durable form, such as on disk, so that they can be recovered in the event of a system crash or failure.

→ For example, in an application that transfers funds from one account to another, the durability property ensures that the changes made to each account will not be reversed.

Transaction States:

1. **Active**

→ The initial state when a transaction is being executed.

2. **Partially Committed**

→ The state after the last operation of a transaction has been executed, but before the transaction is officially committed.

3. **Committed**

→ The state when all operations of a transaction have been successfully completed, and the changes are made permanent.

4. **Aborted**

→ The state when a transaction is rolled back, and any changes made by the transaction are undone.

Serializability

Serializability is a concept in database management systems (DBMS) that ensures the correct and consistent execution of transactions in a multi-user environment. It guarantees that the final result of a concurrent execution of multiple transactions is equivalent to the result that would be obtained if the transactions were executed in some serial order, one after the other. In other words, the system behaves as if transactions were executed in a serial, non-concurrent manner.

Serializability Levels

1. **Serializable (SER)**

→ The highest level of serializability. Enforces the strictest form of consistency, ensuring that the result of concurrent transactions is the same as if they were executed serially.

2. **Serializable Read (SR)**

→ Guarantees serializability for read operations but allows more flexibility for write operations.

3. **Repeatable Read (RR)**

→ Guarantees that each transaction sees a consistent snapshot of the database, preventing the phenomenon of non-repeatable reads.

Serializable Schedule

A serializable schedule in a database refers to a specific order in which a set of transactions can be executed concurrently while preserving the illusion that they are executed in a serial (non-concurrent) manner. The goal is to achieve the highest level of isolation among transactions to prevent issues like lost updates, inconsistent reads, and other anomalies that may arise due to concurrent execution.

Concurrency Control: Need of Concurrency Control, Lock-Based Protocols

Concurrency Control in Database Management System is a procedure of managing simultaneous operations without conflicting with each other. It ensures that Database transactions are performed concurrently and accurately to produce correct results without violating data integrity of the respective Database.

Need for Concurrency Control:

1. **Isolation of Transactions**

→ Concurrent execution of transactions should not lead to interference or influence the outcome of each other. Each transaction should operate on a consistent snapshot of the database.

2. **Consistency**

→ The database should remain in a consistent state, adhering to integrity constraints, even when multiple transactions are executing concurrently.

3. **Avoidance of Data Inconsistency**

→ Conflicting operations, such as read and write operations on the same data by different transactions, can lead to data inconsistency if not properly controlled.

4. **Prevention of Anomalies**

→ Without concurrency control, anomalies such as lost updates, non-repeatable reads, and dirty reads can occur, compromising the reliability of the data.

Lock-Based Protocols:

→ Lock-based protocols are a widely used approach for concurrency control, where transactions request and release locks on data items to control access. Various types of locks can be used, such as shared locks and exclusive locks, to coordinate access to the data.

1. **Shared Lock (S-Lock)**

→ Multiple transactions can acquire shared locks on a data item simultaneously. A shared lock allows a transaction to read the data item but not modify it.

2. **Exclusive Lock (X-Lock)**

→ Only one transaction can acquire an exclusive lock on a data item at a time. An exclusive lock allows a transaction to both read and modify the data item.

3. **Lock-Based Protocols**

A) Two-Phase Locking (2PL)

→ Transactions go through two phases: a growing phase (acquiring locks) and a shrinking phase (releasing locks). Guarantees serializability and avoids conflicts by enforcing certain rules about lock acquisition and release.

B) Strict Two-Phase Locking

→ An extension of 2PL with the additional rule that no locks can be released until the transaction is committed.Ensures a higher level of isolation but may lead to reduced concurrency.

4. Timestamp Ordering

→ Assigns a unique timestamp to each transaction and orders transactions based on their timestamps. Conflicts are resolved by comparing timestamps, and transactions are allowed to proceed in timestamp order.

5. Multiple Granularity Locking

→ Allows locks to be acquired at different levels of granularity (e.g., at the level of a data item, page, or table). Helps in minimizing contention and improving concurrency.

Recovery: Failure Classification, Shadow paging

→ Recovery in database management systems refers to the process of restoring the database to a consistent and correct state after a failure. Failures can occur due to various reasons, including hardware malfunctions, software errors, or other unforeseen events. Recovery mechanisms are designed to handle these failures and ensure data integrity. One recovery technique is shadow paging.

Failure Classification

Failures in a database system can be classified into different categories:

1. Transaction Failures

→ Failures that occur during the execution of a transaction. Can be transient (temporary) or permanent.

2. System Failures

→ Failures that affect the entire database system. Can be due to hardware faults, power outages, software bugs, etc.

3. Media Failures:

→ Failures that result in the loss of data stored on a storage medium (disk failure, file corruption, etc.).

Shadow Paging

- Shadow paging is a recovery technique that involves creating a shadow or duplicate copy of the entire database before any modifications are made. The original database is referred to as the "current page," and the duplicate is referred to as the "shadow page." The idea is to perform all modifications on the shadow page, and once a transaction is committed, the shadow page becomes the new current page.

Steps in Shadow Paging

1. **Initialization**

→ Create a duplicate copy of the entire database, known as the shadow page. Maintain a page table to track the mapping between the current page and the shadow page.

2. **Transaction Execution**

→ All modifications are made on the shadow page. If a transaction aborts, the changes are discarded by reverting to the original state in the current page.

3. **Committing a Transaction**

→ Update the page table to reflect the new mapping between the current page and the shadow page. Make the shadow page the new current page.

4. **Recovery After a System Failure**

→ If a system failure occurs, the database can be recovered by using the page table to identify the consistent state. The shadow page can be copied back to the current page.