Advance Database Concepts

UNIT-1

Transaction Management:

Transaction management refers to the process of ensuring data integrity and consistency in a database by executing multiple database operations as a single logical unit of work.

It involves the management of the lifecycle of a database transaction, including the initiation, execution, and completion of transactions.

The following are some basic concepts related to transaction management:

- ***Transaction:*** A transaction is a sequence of one or more database operations that are executed as a single logical unit of work. A transaction must be atomic, consistent, isolated, and durable (ACID) to ensure data integrity.
- ***Commit:*** When a transaction is successfully completed, it must be committed to make the changes permanent in the database.
- ***Rollback***: If a transaction encounters an error or failure, it must be rolled back to its original state before the transaction started to maintain data consistency.
- ***Isolation:*** Isolation refers to the degree to which the effects of one transaction are visible to other transactions that are executing concurrently.
- ***Concurrency control:*** Concurrency control is the process of managing multiple transactions that are executing concurrently to ensure data consistency and integrity.
- ***Deadlock:*** A deadlock occurs when two or more transactions are waiting for each other to release resources, resulting in a cycle of waiting that cannot be resolved without intervention.
- ***Savepoints***: Savepoints allow a transaction to be divided into smaller units and provide a way to rollback a portion of a transaction without rolling back the entire transaction.

Transaction States:

A transaction goes through several states as it executes. Understanding these states is essential to ensure the proper management of transactions. The following are the different states of a transaction:

- ***Active:*** When a transaction begins, it enters the active state. In this state, the transaction is executing, and it can issue read and write operations on the database.

- ***Partially Committed***: After executing all the database operations of a transaction, the transaction enters the partially committed state. In this state, the transaction has executed successfully, but the changes made by the transaction are not yet permanent. The transaction can still be rolled back.

- ***Committed:*** After the transaction has been partially committed, it enters the committed state. In this state, the transaction's changes are made permanent in the database, and the transaction is completed.

- ***Aborted***: If a transaction fails to execute or encounters an error, it enters the aborted state. In this state, the transaction's changes are undone, and the database is restored to its original state before the transaction began.

- ***Failed*****:** A transaction can fail to execute for various reasons, such as a deadlock or an exception. In this state, the transaction has not started, and it is not considered active.

- ***In-doubt***: In-doubt transactions occur in distributed databases when the coordinator node has not received confirmation of the transaction's status from all the participating nodes. In this state, the transaction is neither committed nor aborted. The transaction remains in this state until the coordinator node receives confirmation from all the participating nodes.

ACID Properties:

ACID is an acronym that stands for Atomicity, Consistency, Isolation, and Durability, which are the four fundamental properties of a transaction in a database. Here are detailed notes on the ACID properties:

- **Atomicity**: Atomicity refers to the property of a transaction where all the operations within a transaction are treated as a single, indivisible unit. A transaction must either complete successfully, or it must be completely undone, and there should be no partial results left behind. If an error occurs during the execution of a transaction, all the changes made within that transaction are rolled back, ensuring that the database remains consistent.

- **Consistency:** Consistency refers to the property of a transaction that ensures that the database remains in a consistent state before and after the transaction executes. This means that the transaction should only take the database from one valid state to another valid state. Any changes made by the transaction must satisfy the integrity constraints and business rules of the database.

- **Isolation:** Isolation refers to the property of a transaction that ensures that it is executed independently of other transactions. This means that the changes made by one transaction should be invisible to other transactions until the transaction is committed. Isolation is necessary to prevent concurrent transactions from interfering with each other and causing inconsistent results.

- **Durability:** Durability refers to the property of a transaction that ensures that the changes made by a transaction are permanent and will survive any subsequent failures. Once a transaction is committed, the changes made within the transaction should be persisted to the database and should be recoverable in the event of a system failure.

Storage Structure :

Storage structure refers to the way data is organized and stored in a database. The type of storage structure used affects the efficiency and speed of data retrieval and manipulation.

Types of storage structures:

- ***Heap File Organization:*** In this type of storage structure, data is stored as an unsorted collection of records. Each record is inserted into the file as it arrives, and there is no particular order to the placement of the records. Retrieval of data in heap file organization is slow because the entire file must be searched to find a particular record.
- ***Sequential*** File Organization: In this type of storage structure, data is stored in a particular order based on a key field. Records are stored in the order of the key field value. Retrieval of data is faster than in heap file organization because the search can be stopped as soon as the required record is found.
- ***Indexed Sequential Access Method (ISAM):*** This storage structure combines the advantages of sequential file organization and indexing. In this method, data is stored in a particular order based on a key field, and an index is created for faster access to the records. The index allows for quick searching for a particular record.
- ***B+ Tree Indexing***: This storage structure is used for large databases. In this method, data is stored in a B+ tree index, which is a balanced tree data structure. The tree is divided into nodes, with the root node containing pointers to the other nodes. The leaf nodes contain the actual data. The B+ tree index is efficient for both retrieval and insertion of data.
- ***Hashing:*** In this storage structure, data is stored in a hash table. The hash function is used to determine the location of the data in the table. The hash function is designed to minimize collisions, which occur when two different pieces of data are mapped to the same location. Hashing is efficient for retrieval of data, but not for insertion or deletion of data.
- ***Clustered Indexing:*** This storage structure is used for tables with a large number of rows. In this method, the table is physically sorted based on the clustered index column. The clustered index column is also used as the primary key. The primary key is used to identify unique rows in the table. Clustered indexing is efficient for both retrieval and insertion of data.

Concurrent Executions

Concurrent execution refers to the ability to execute multiple transactions simultaneously in a database management system (DBMS). Concurrent execution provides users with better performance and allows multiple users to access the database at the same time. However, it also introduces several challenges related to data consistency and integrity.

Concurrency control is the process of managing multiple transactions that are executing concurrently to ensure data consistency and integrity. There are several techniques used for concurrency control, including locking, timestamp ordering, and optimistic concurrency control.

Locking is a popular technique used for concurrency control. Locks are used to control access to resources and ensure that only one transaction can access a resource at a time. There are two types of locks used in locking: shared locks and exclusive locks. Shared locks allow multiple transactions to read a resource simultaneously, while exclusive locks allow only one transaction to update or delete a resource.

Timestamp ordering is another technique used for concurrency control. Each transaction is assigned a unique timestamp, and the order of execution is determined based on the timestamp. The oldest transaction is executed first, and transactions with a newer timestamp are executed in a later order.

Optimistic concurrency control is a technique that assumes that conflicts between transactions are rare. Transactions are allowed to execute concurrently without acquiring locks, and any conflicts are detected and resolved during the commit phase. If a conflict is detected, one of the transactions is rolled back and restarted.

Serializability:

Serializability is a concept in transaction management that ensures that a set of concurrent transactions produce the same result as if they had been executed serially (one after the other). It is a property of a schedule, which is a sequence of actions (operations) that occur in a database system over time.

Serializability ensures that concurrent transactions do not interfere with each other and that the end result is consistent with the outcome that would have occurred if the transactions had been executed in a serial order. This is important in multi-user database systems where multiple users may be accessing the same data at the same time.

There are two types of serializability:

Conflict Serializability: In conflict serializability, transactions are executed in a way that no two transactions conflict with each other. The main focus of this type of serializability is on ensuring that the operations of two transactions that access the same data item do not overlap. The order of execution of transactions in conflict serializability should result in the same database state as any other possible order of the same transactions. The conflict graph is used to determine the conflict serializability of transactions.

View Serializability: In view serializability, transactions are executed in a way that ensures the same results as if the transactions were executed serially in some order. In view serializability, the transactions

are not necessarily independent of each other. Instead, they can access the same data items and perform operations that depend on the results of the previous transactions.

To ensure serializability, the database system uses concurrency control mechanisms such as locking, timestamping, and validation to prevent conflicting transactions from accessing the same data at the same time. By ensuring that transactions are serialized in a consistent manner, serializability guarantees that the database system is reliable and consistent, and that data integrity is maintained.

Recoverability:

Recoverability is a property of a transaction that ensures that a transaction can be recovered in the event of a system failure. In recoverable transactions, if a transaction has made changes to the database, those changes must be written to disk before the transaction is committed. This ensures that the changes made by the transaction can be recovered in the event of a system failure.

In recoverable transactions, if a transaction fails, any changes made by the transaction must be undone or rolled back. This ensures that the database remains consistent and that any changes made by the transaction are not lost.

Concurrency control

Concurrency control is an essential concept in transaction management, which is used to manage multiple transactions executing concurrently. It ensures that the database remains consistent and the data integrity is maintained. In a multi-user database system, multiple transactions may try to access and modify the same data simultaneously. If the transactions are not managed properly, they may result in a conflict and data inconsistency.

Lock-based protocols

Lock-based protocols are used to manage concurrency in databases. In this approach, transactions acquire locks on the data items they need to access to ensure that other transactions do not modify or read them at the same time. Lock-based protocols use various modes of locks to provide different levels of concurrency and data integrity.

The following are the different modes of locks used in lock-based protocols:

- ***Shared (S) Locks***: A shared lock allows multiple transactions to read the same data item simultaneously but prevents them from modifying it. A shared lock is also known as a read lock, as it allows only read operations.
- ***Exclusive (X) Locks:*** An exclusive lock allows a transaction to both read and modify a data item. Only one transaction can hold an exclusive lock on a data item at a time.
- ***Update (U) Locks:*** An update lock is a combination of a shared and exclusive lock. It is used when a transaction wants to read a data item and update it later in the transaction. An update lock

prevents other transactions from acquiring exclusive locks on the same data item but allows multiple transactions to acquire shared locks.

- ***Intent (I) Locks:*** An intent lock is used to indicate the intention of a transaction to acquire a higher-level lock on a data item. For example, when a transaction acquires an exclusive lock, it must first acquire an intent exclusive lock on the entire object, indicating that it will be acquiring an exclusive lock on the object. This ensures that no other transaction can acquire any conflicting locks on the object.

- ***Shared and Update (SU) Locks***: An SU lock is a combination of a shared and update lock. It is used when a transaction wants to read a data item and update it later in the transaction, but it allows other transactions to also read the same data item concurrently.

- ***Shared Intent (SI) Locks:*** A shared intent lock is used to indicate the intention of a transaction to acquire shared locks on a range of data items. This lock is used to optimize the locking process by reducing the number of locks that need to be acquired.

Granting of Locks:

The process of granting locks involves the following steps:

- ***Requesting a lock***: A transaction requests a lock on a data item by sending a request to the database management system.

- ***Checking for conflicts:*** The database management system checks if the requested lock conflicts with any existing locks held by other transactions. If there is a conflict, the requesting transaction is either blocked or rolled back.

- ***Granting the lock:*** If there is no conflict, the database management system grants the requested lock to the transaction.

- ***Releasing the lock:*** Once the transaction completes its operation on the data item, it releases the lock to allow other transactions to access the data.

The granting of locks is an important aspect of transaction management that ensures the integrity and consistency of data in a database. It is important to ensure that locks are granted and released correctly to prevent deadlock and ensure that all transactions can access and modify data as needed.

The Two-Phase Locking (2PL):

The Two-Phase Locking (2PL) protocol is a concurrency control mechanism used to ensure data consistency and prevent transaction conflicts in database systems. The protocol is based on the principles of locking and provides strict guarantees on transaction serialization and conflict resolution.

The 2PL protocol works in two phases, as follows:

- ***Growing Phase:*** In this phase, the transaction acquires locks on the required resources, one at a time. Once a lock is acquired, it is not released until the transaction either commits or aborts. During this phase, the transaction can only acquire locks, but cannot release any locks.

- ***Shrinking Phase***: In this phase, the transaction releases the locks on the acquired resources, one at a time. Once a lock is released, it cannot be re-acquired. During this phase, the transaction can only release locks, but cannot acquire any new locks.

The 2PL protocol guarantees the following properties:

- ***Serializability***: The protocol ensures that transactions are executed in a serializable order, meaning that the outcome of the concurrent execution is equivalent to some serial execution of the same transactions.

- ***Deadlock avoidance:*** The protocol prevents deadlocks from occurring by ensuring that transactions acquire locks in a predetermined order and by releasing all locks at the end of the transaction.

- ***Consistency***: The protocol ensures that the database remains consistent by preventing transactions from accessing the same data simultaneously and by enforcing the ACID properties of transactions.

Time Stamp Based Protocols:

Time stamp based protocols are used to ensure the serializability of transactions in a database. These protocols use time stamps to order the execution of transactions and to prevent conflicts between transactions that are executing concurrently. Two popular time stamp based protocols are the Timestamp-Ordering Protocol and Thomas' Write Rule.

Timestamp-Ordering Protocol:

The Timestamp-Ordering Protocol assigns a unique timestamp to each transaction and ensures that transactions are executed in the order of their timestamps. If a transaction tries to read or write data that has been modified by a transaction with a higher timestamp, the lower timestamp transaction is rolled back. The protocol uses the following rules to ensure serializability:

- ***Write operation rule***: A transaction with a higher timestamp can write to a data item only if no transaction with a lower timestamp has already written to the same data item.

- ***Read operation rule***: A transaction with a higher timestamp can read a data item modified by a transaction with a lower timestamp.

- ***Abort rule:*** A transaction with a lower timestamp is aborted if it tries to read or write a data item modified by a transaction with a higher timestamp.

Thomas' Write Rule:

Thomas' Write Rule is a time stamp based protocol that prevents write-write conflicts between transactions. The protocol uses a timestamp ordering scheme and the following rules:

- A transaction can write to a data item only if its timestamp is greater than the timestamp of the last transaction that wrote to that data item.
- If a transaction with a lower timestamp tries to write to a data item, it is aborted.

Validation based Protocols:

Validation based protocols are used to ensure the serializability of transactions in a database. These protocols use a validation phase to check whether the execution of a transaction will result in a serializable schedule. Two popular validation based protocols are the Validation Protocol and the Optimistic Concurrency Control Protocol.

Validation Protocol:

The Validation Protocol uses a validation phase to check whether the execution of a transaction will result in a serializable schedule. The protocol uses the following steps:

- Each transaction is assigned a timestamp.
- When a transaction wants to commit, it enters the validation phase.
- The protocol checks whether the execution of the transaction will result in a serializable schedule.
- If the schedule is serializable, the transaction is committed. If not, it is rolled back.

Optimistic Concurrency Control Protocol:

The Optimistic Concurrency Control Protocol assumes that conflicts between transactions are rare and allows transactions to execute concurrently without locking. The protocol uses the following steps:

1. Each transaction is assigned a timestamp.
2. When a transaction wants to commit, it enters the validation phase.
3. The protocol checks whether the execution of the transaction will result in a serializable schedule.
4. If the schedule is serializable, the transaction is committed. If not, it is rolled back.
5. If conflicts occur, the protocol rolls back one or more transactions and retries the transaction that caused the conflict.

Deadlock handling

Deadlock handling is the process of resolving deadlocks in a database system. Deadlocks occur when two or more transactions are waiting for each other to release resources, resulting in a cycle of waiting that cannot be resolved without intervention.

The following are the steps involved in deadlock handling:

1. **Detection:** The first step in deadlock handling is to detect the existence of a deadlock. This can be done using various algorithms, such as wait-for graph or timeout detection.
2. **Assessment:** Once a deadlock is detected, the system must assess the severity of the deadlock and decide which transactions to terminate or rollback. This decision is based on various factors, such as the priority of transactions, the amount of work already completed by the transactions, and the resources that are currently being held by the transactions.
3. **Resolution:** The next step is to resolve the deadlock by terminating or rolling back one or more transactions. The system may choose to terminate the transaction that has completed the least amount of work or the transaction that has the lowest priority.
4. **Recovery:** After the deadlock is resolved, the system must recover from the deadlock by releasing the resources that were held by the terminated transactions and by restarting any transactions that were rolled back.

Methods that can be used to handle deadlocks:

1. ***Timeout:*** The system can set a timeout value for each transaction and terminate any transaction that exceeds its timeout value.
2. ***Resource allocation***: The system can allocate resources in a way that prevents deadlocks from occurring. For example, the system can ensure that all transactions request resources in the same order.
3. ***Deadlock detection and resolution***: The system can detect deadlocks and resolve them using one of the algorithms mentioned above.
4. ***Prevention:*** The system can prevent deadlocks by ensuring that transactions do not request resources that are already held by other transactions.

Deadlock Prevention:

Deadlock prevention is a method of avoiding deadlocks in a database system. It involves designing the system in such a way that it is impossible for a deadlock to occur. There are several techniques that can be used for preventing deadlocks. The following are some of the techniques used in deadlock prevention:

- ***Lock ordering:*** This technique involves enforcing a strict order in which locks can be obtained. For example, if a transaction needs to obtain a lock on resource A before resource B, then all transactions must follow the same order. This method ensures that deadlocks cannot occur due to conflicting lock requests.

- ***Timeouts:*** This technique involves setting a timeout for each lock request. If a lock is not granted within the specified time, the transaction is aborted, and the resources are released. This ensures that transactions do not wait indefinitely for a lock to be granted.

- ***Resource allocation***: This technique involves allocating resources to transactions only if all the resources required by the transaction are available. This ensures that a transaction cannot acquire some resources and then wait for other resources that are already being held by another transaction.

- ***Two-phase locking:*** This technique involves dividing the transaction into two phases: the locking phase and the unlocking phase. During the locking phase, a transaction acquires all the required locks before releasing any locks. During the unlocking phase, a transaction releases all the locks it has acquired. This method ensures that deadlocks cannot occur due to conflicting lock requests.

- ***Avoiding circular waits***: This technique involves ensuring that transactions do not form a circular wait for resources. For example, if transaction 1 is waiting for a resource held by transaction 2, then transaction 2 cannot wait for a resource held by transaction 1. This ensures that circular waits cannot occur.

Deadlock Avoidance

Deadlock avoidance is a strategy used in transaction management to prevent the occurrence of deadlocks. Deadlocks occur when two or more transactions are waiting for each other to release resources, which results in a cycle of waiting that cannot be resolved without intervention. Deadlock avoidance works by preventing transactions from acquiring locks that could lead to a deadlock situation.

There are several techniques used in deadlock avoidance, including:

Two-phase locking: Two-phase locking is a technique that ensures serializability by having a transaction acquire all the locks it needs before it executes any updates, and hold onto these locks until the transaction is complete. This technique prevents deadlocks by ensuring that transactions cannot request additional locks that could cause a deadlock.

Timeouts: Timeouts are used to prevent transactions from waiting indefinitely for resources. A timeout mechanism can be used to limit the amount of time a transaction can wait for a lock or a resource. If a transaction cannot obtain a lock or resource within the specified time limit, it is rolled back, and its resources are released.

Lock ordering: Lock ordering is a technique that involves acquiring locks in a specific order to prevent deadlocks. This technique can be used when transactions need to acquire multiple locks to complete their tasks. By acquiring locks in a specific order, transactions can avoid waiting for locks that other transactions are holding onto.

Resource allocation graphs: Resource allocation graphs are used to determine if a deadlock is possible. Each transaction is represented as a node in the graph, and each resource is represented as an edge between the transactions that are using the resource. By analyzing the graph, it is possible to determine if a deadlock is possible and take corrective measures to avoid it.

Deadlock Detection:

Deadlock is a situation in a database system where two or more transactions are waiting for each other to release the resources that they hold, resulting in a cycle of waiting that cannot be resolved without intervention. Deadlock detection is the process of identifying and resolving deadlocks in a database system. The following steps are involved in deadlock detection:

- ***Wait-for graph creation:*** The first step in deadlock detection is to create a wait-for graph. The wait-for graph is a directed graph that represents the dependencies between transactions.
- ***Cycle detection***: Once the wait-for graph is created, the next step is to check if there is a cycle in the graph. If there is a cycle, it means that a deadlock has occurred.
- ***Deadlock resolution:*** If a deadlock is detected, there are several ways to resolve it. One way is to rollback one of the transactions involved in the cycle, which will release the resources held by that transaction and allow the other transactions to proceed. Another way is to abort all the transactions involved in the cycle and start them again from the beginning.

Deadlock Recovery:

Deadlock recovery is the process of recovering from a deadlock in a database system. The following methods can be used for deadlock recovery:

Rollback: Rollback is the most common method of deadlock recovery. In this method, one of the transactions involved in the deadlock is rolled back, which releases the resources held by that transaction and allows the other transactions to proceed.

Abort: In some cases, it may be necessary to abort all the transactions involved in the deadlock and start them again from the beginning.

Timeouts: Another method of deadlock recovery is to set timeouts on transactions. If a transaction does not complete within the specified time, it is rolled back, which releases the resources held by that transaction and allows other transactions to proceed.

Resource allocation: Deadlocks can be prevented by ensuring that resources are allocated in a way that prevents the possibility of a deadlock. This can be done by using locking and scheduling algorithms that ensure that transactions do not wait for resources that are held by other transactions.

CRAFTED BY:CODECHAMP