UNIVERSITY OF GHANA, LEGON

(All rights reserved)

DEPARTMENT OF COMPUTER ENGINEERING
SCHOOL OF ENGINEERING
SEMESTER 1 2022/2023 ACADEMIC YEAR

Database Group Project Documentation

Course Code /Title: CPEN 211 Database System Design
Credits: 3

GROUP 2

Members

MICHELLE OWUSU - 10957340

APPIAH YAW FRIMPONG - 10987818

MENSAH NYANYO HUBERT - 10976127

BENTIL B. REXFORD - 10946257

ANANE GEORGE NYARKO - 10947340

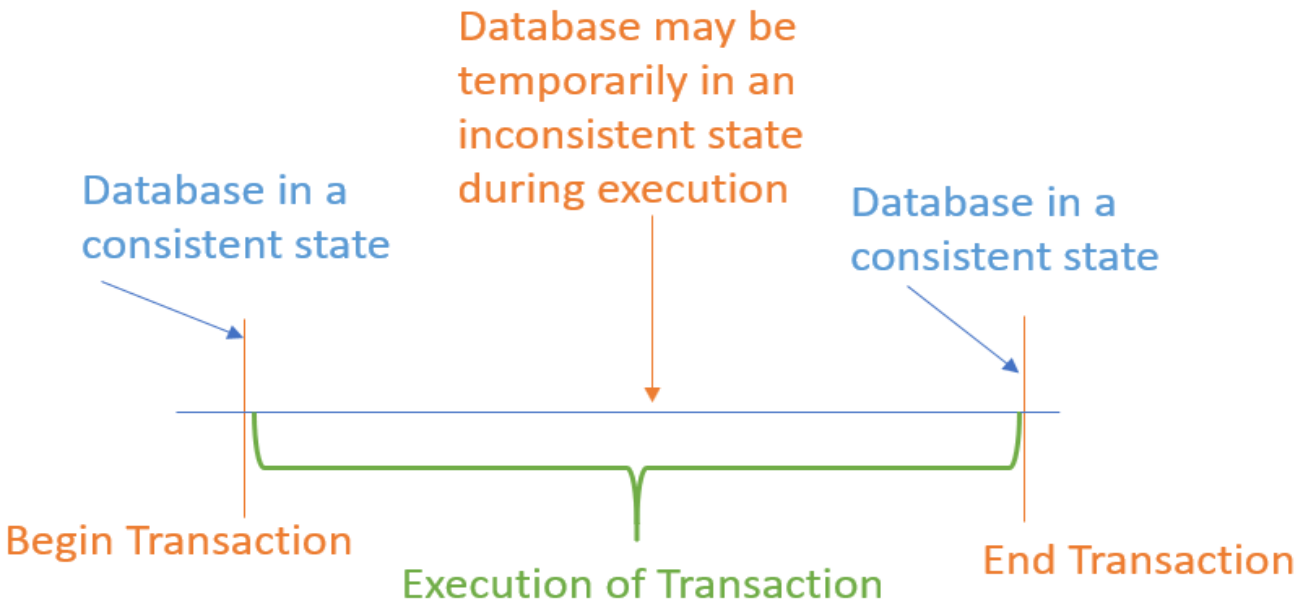
EVANS ACHEAMPONG - 10987644

DERY-KUUZUME SANDRA - 10986424

TRANSACTION MANAGEMENT IN DATABASE DESIGN

Abstract

Transaction management is a fundamental aspect of database design, which involves managing a group of database operations as a single logical unit. In a transaction, a sequence of database operations are executed either fully or not at all, ensuring the consistency and integrity of data. The transaction management subsystem in a database management system (DBMS) is responsible for ensuring that transactions are atomic, consistent, isolated, and durable (ACID), and for handling concurrent transactions in a multi-user environment. This abstract provides an overview of the principles of transaction management, including the use of transaction logs, lock-based concurrency control, and deadlock detection and resolution. It also discusses the various techniques used to optimize transaction processing, such as commit and rollback protocols, deferred updates, and optimistic concurrency control. By implementing transaction management effectively, database designers can ensure the reliability and performance of their databases, while minimizing the risk of data loss or corruption.



Introduction

In today's data-driven world, databases are an essential component of many software applications, from enterprise resource planning systems to e-commerce websites. Ensuring the reliability, consistency, and integrity of the data stored in these databases is critical to the success of these applications. Transaction management is a key aspect of database design that plays a crucial role in achieving these goals.

A transaction is a logical unit of work that consists of one or more database operations, such as insert, update, or delete. In a well-designed database system, transactions are executed as atomic, consistent, isolated, and durable (ACID) operations, ensuring the integrity and consistency of the database. For example, consider a banking system that allows customers to transfer funds between accounts. If the system were not designed to handle transactions properly, it could result in the loss of funds or other errors.

The transaction management subsystem in a database management system (DBMS) is responsible for ensuring that transactions are executed correctly, despite failures or concurrency issues that may arise in a multi-user environment. This subsystem uses techniques such as transaction logs, lock-based concurrency control, and deadlock detection and resolution to manage transactions effectively.

Optimizing transaction processing is another important aspect of transaction management. Techniques such as commit and rollback protocols, deferred updates, and optimistic concurrency control can help improve the performance and scalability of a database system.

This paper provides an overview of transaction management in database design, including its principles, techniques, and best practices. It discusses the importance of transaction management in ensuring the reliability and consistency of database systems, and provides guidance on how to design and optimize transaction processing for various applications. By understanding and implementing transaction management effectively,

database designers can create robust, reliable, and high-performance database systems that meet the needs of their users.

Facts about Database Transactions

- A transaction is a program unit whose execution may or may not change the contents of a database.
- The transaction concept in DBMS is executed as a single unit.
- If the database operations do not update the database but only retrieve data, this type of transaction is called a read-only transaction.
- A successful transaction can change the database from one CONSISTENT STATE to another
- DBMS transactions must be atomic, consistent, isolated and durable
- If the database were in an inconsistent state before a transaction, it would remain in the inconsistent state after the transaction.

Importance of concurrency in Transactions

A database is a shared resource accessed. It is used by many users and processes concurrently. For example, the banking system, railway, and air reservations systems, stock market monitoring, supermarket inventory, and checkouts, etc.

Not managing concurrent access may create issues like:

- Hardware failure and system crashes

- Concurrent execution of the same transaction, deadlock, or slow performance.

Examples of how transaction management is used in database design

1. Banking Transactions: In a banking system, transaction management is used to ensure that money transfers between accounts are completed successfully, and that no funds are lost or duplicated. For example, if a customer transfers money from their checking account to their savings account, the transaction must be executed as a single unit of work, so that the funds are either transferred in full or not at all.
2. Online Purchases: In an e-commerce system, transaction management is used to ensure that orders are processed correctly, and that inventory levels and financial records are updated accurately. For example, when a customer places an order, the system must execute a series of database operations, such as updating the inventory and processing the payment, as a single transaction.
3. Healthcare Records: In a healthcare system, transaction management is used to ensure that patient records are updated accurately and securely. For example, when a doctor updates a patient's medical record, the changes must be executed as a single transaction, so that the record remains consistent and secure.
4. Flight Reservation System: In a flight reservation system, transaction management is used to ensure that seats are reserved correctly, and that no double bookings occur. For example, when a customer makes a reservation, the system must execute a series of database operations, such as updating the seat availability and reserving the seat, as a single transaction.
5. Social Media: In a social media system, transaction management is used to ensure that user interactions, such as likes, comments, and shares, are recorded accurately and consistently. For example, when a user likes a post, the system must execute a series of database operations, such as updating the user's activity log and updating the post's likes count, as a single transaction.

Principles of Transaction Management (ACID Properties)

A transaction is a logical unit of work that consists of one or more database operations, such as insert, update, or delete. Transactions are executed as **A**tomic, **C**onsistent, **I**solated, and **D**urable (ACID) operations, ensuring the integrity and consistency of the database. Let's take a closer look at each of these principles:

- **Atomicity**: A transaction is executed as a single unit of work, so that all of its operations are either completed successfully, or the transaction is rolled back, and no changes are made to the database.
- **Consistency**: A transaction ensures that the database remains consistent before and after its execution, regardless of any failures or concurrency issues that may arise during its execution.
- **Isolation**: A transaction is executed independently of other transactions, so that each transaction sees a consistent snapshot of the database, without being affected by the changes made by other transactions.
- **Durability**: Once a transaction is committed, its changes become permanent, and are not lost even in the event of system failures or crashes.

ACID Property in DBMS with example:

Below is an example of ACID property in DBMS:

Transaction 1: Begin X=X+50, Y = Y-50 END
Transaction 2: Begin X=1.1*X, Y=1.1*Y END

Transaction 1 is transferring $50 from account X to account Y.

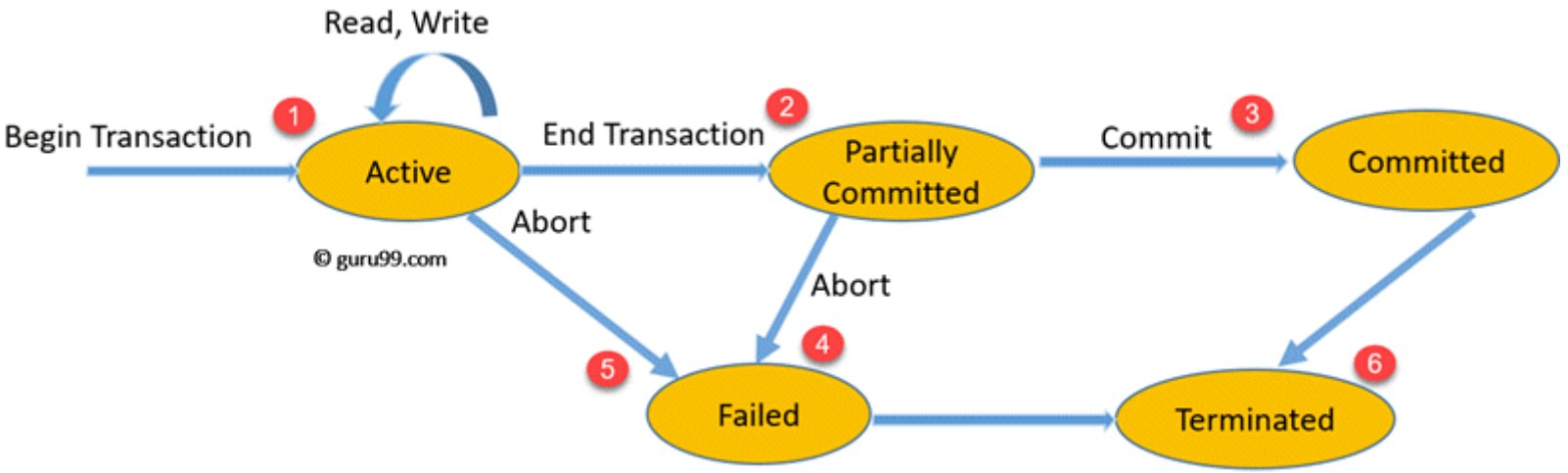
Transaction 2 is crediting each account with a 10% interest payment.

If both transactions are submitted together, there is no guarantee that the Transaction 1 will execute before Transaction 2 or vice versa. Irrespective of the order, the result must be as if the transactions take place serially one after the other.

States of Transactions

The various states of a transaction concept in DBMS are listed below:

State	Transaction types
Active State	A transaction enters into an active state when the execution process begins. During this state read or write operations can be performed.
Partially Committed	A transaction goes into the partially committed state after the end of a transaction.
Committed State	When the transaction is committed to state, it has already completed its execution successfully. Moreover, all of its changes are recorded to the database permanently.
Failed State	A transaction considers failed when any one of the checks fails or if the transaction is aborted while it is in the active state.
Terminated State	State of transaction reaches terminated state when certain transactions which are leaving the system can’t be restarted



Let's study a [state transition diagram] that highlights how a transaction moves between these various states.

1. Once a transaction states execution, it becomes active. It can issue READ or WRITE operation.
2. Once the READ and WRITE operations complete, the transactions becomes partially committed state.
3. Next, some recovery protocols need to ensure that a system failure will not result in an inability to record changes in the transaction permanently. If this check is a success, the transaction commits and enters into the committed state.
4. If the check is a fail, the transaction goes to the Failed state.
5. If the transaction is aborted while it's in the active state, it goes to the failed state. The transaction should be rolled back to undo the effect of its write operations on the database.
6. The terminated state refers to the transaction leaving the system.

Techniques of Transaction Management

The transaction management subsystem in a database management system (DBMS) is responsible for ensuring that transactions are executed correctly, despite failures or concurrency issues that may arise in a multi-user environment. Let's take a look at some of the techniques used by the transaction management subsystem:

- Transaction Logs: A transaction log records all the changes made to the database during the execution of a transaction. In the event of a failure or crash, the transaction log can be used to restore the database to a consistent state.
- Lock-Based Concurrency Control: In a multi-user environment, transactions can access the same data concurrently, resulting in conflicts and inconsistencies. Lock-based concurrency control ensures that only one transaction can modify a piece of data at a time, by acquiring and releasing locks on the data.
- Deadlock Detection and Resolution: Deadlocks occur when two or more transactions are blocked, waiting for locks held by each other. Deadlock detection and resolution techniques ensure that deadlocks are detected and resolved, so that the transactions can proceed.

Best Practices for Transaction Management

To ensure effective transaction management, database designers should follow best practices such as:

- Minimize the length of transactions: Shorter transactions reduce the risk of deadlocks, and make it easier to roll back changes in the event of a failure.
- Use optimistic concurrency control: Optimistic concurrency control allows multiple transactions to access the same data concurrently, by allowing each transaction to proceed without acquiring locks, and checking for conflicts before committing the changes.
- Use commit and rollback protocols: Commit protocols ensure that all the changes made by a transaction are committed together, while rollback protocols ensure that all the changes made by a transaction are rolled back together.
- Monitor and tune transaction performance: Monitoring and tuning transaction performance can help improve the scalability and reliability of the database system, by identifying and resolving performance bottlenecks.

Disadvantages of using a Transaction

- It may be difficult to change the information within the transaction database by end-users.
- We need to always roll back and start from the beginning rather than continue from the previous state.

Types of Transactions

Based on Application areas

- Non-distributed vs. distributed
- Compensating transactions
- Transactions Timing
- On-line vs. batch

Based on Actions

- Two-step
- Restricted
- Action model

Based on Structure

- Flat or simple transactions: It consists of a sequence of primitive operations executed between a begin and end operations.
- Nested transactions: A transaction that contains other transactions.
- Workflow

Schedules

A Schedule is a process creating a single group of the multiple parallel transactions and executing them one by one. It should preserve the order in which the instructions appear in each transaction. If two transactions are executed at the same time, the result of one transaction may affect the output of other.

Example

Initial Product Quantity is 10
Transaction 1: Update Product Quantity to 50
Transaction 2: Read Product Quantity

If Transaction 2 is executed before Transaction 1, outdated information about the product quantity will be read. Hence, schedules are required.

Parallel execution in a database is inevitable. But, Parallel execution is permitted when there is an equivalence relation amongst the simultaneously executing transactions. This equivalence is of 3 Types.

RESULT EQUIVALENCE

If two schedules display the same result after execution, it is called result equivalent schedule. They may offer the same result for some value and different results for another set of values. For example, one transaction updates the product quantity, while other updates customer details.

View Equivalence

View Equivalence occurs when the transaction in both the schedule performs a similar action. Example, one transaction inserts product details in the product table, while another transaction inserts product details in the archive table. The transaction is the same, but the tables are different.

CONFLICT Equivalence

In this case, two transactions update/view the same set of data. There is a conflict amongst transaction as the order of execution will affect the output.

Serializability

Serializability is the process of search for a concurrent schedule who output is equal to a serial schedule where transaction is execute one after the other. Depending on the type of schedules, there are two types of serializability:

- Conflict
- View

Summary

- Transaction management is a logical unit of processing in a DBMS which entails one or more database access operation
- It is a transaction is a program unit whose execution may or may not change the contents of a database.
- Not managing concurrent access may create issues like hardware failure and system crashes.
- Active, Partially Committed, Committed, Failed & Terminate are important transaction states.
- The full form of ACID Properties in DBMS is Atomicity, Consistency, Isolation, and Durability
- Three DBMS transactions types are Base on Application Areas, Action, & Structure.
- A Schedule is a process creating a single group of the multiple parallel transactions and executing them one by one.
- Serializability is the process of search for a concurrent schedule whose output is equal to a serial schedule where transactions are executed one after the other.

Conclusion

Transaction management is a critical aspect of database design that plays a crucial role in ensuring the reliability, consistency, and integrity of the data stored in databases. By understanding and implementing transaction management effectively, database designers can create robust, reliable, and high-performance database systems that meet the needs of their users.

References

[1] **https://www.guru99.com/dbms-transaction-management.html**

[2] **https://www.geeksforgeeks.org/transaction-management/**