KARNAVATI UNIVERSITY | UIT

Unitedworld Institute Of Technology

B.Tech. Computer Science & Engineering

Semester : 3rd

Introduction To Database Management System
Course Code: 71203002003

Unit IV : TRANSACTIONS, CONCURRENCY CONTROL AND DEADLOCKS

Prepared by:
Mr. Utsav Kapadiya
Assistant Professor (UIT)

Understanding Database Transactions

A transaction is a fundamental concept in database management systems that ensures data integrity and reliability. It represents a group of tasks that form one complete operation in a database. The critical principle: if any one task fails, the whole transaction fails and must be rolled back to maintain data accuracy.

Real-World Example: Money Transfer

The Scenario

When a bank employee transfers ₹500 from Account A to Account B, what appears as a simple operation actually involves multiple critical steps that must all succeed together.

This seemingly straightforward transaction demonstrates why databases need sophisticated mechanisms to ensure data integrity and prevent financial discrepancies.

The Complete Transaction Process

1

2

Account A Operations

1. Open A's account
2. Read old balance
3. Subtract ₹500
4. Save new balance
5. Close A's account

Account B Operations

1. Open B's account
2. Read old balance
3. Add ₹500
4. Save new balance
5. Close B's account

Critical Point: If *any step fails* (like a power cut before saving B's balance), the transaction must undo everything to keep data correct. This is why transaction management is essential for database reliability.

ACID Properties: The 4 Golden Rules

Each transaction must follow ACID principles to keep the database reliable and trustworthy. These four properties work together to ensure data integrity, consistency, and durability across all database operations.

A – Atomicity: All or Nothing

Core Principle

Either all steps of a transaction are completed successfully, or none of them are applied to the database.

Protection Mechanism

If ₹500 is deducted from Account A but not added to Account B due to an error, the system will undo the deduction from A.

Result

No money is lost or created. The database returns to its original state as if the transaction never occurred.

C – Consistency: Maintaining Valid Data

The Principle

Data must always remain valid and logical. All database rules, constraints, and relationships must be preserved before and after every transaction.

Example in Action

Before and after the money transfer transaction, the total money in Account A + Account B should remain the same. No rules of the database should be broken.

₹7000

Before Transaction

A: ₹5000 + B: ₹2000

₹7000

After Transaction

A: ₹4000 + B: ₹3000

The total remains constant, proving the database maintained consistency throughout the operation.

I – Isolation: Independent Execution

Meaning

Each transaction should act like it's the only one running, even when multiple transactions execute simultaneously.

Real-World Scenario

If two people are transferring money at the same time, their transactions shouldn't mix up and cause errors. They run *as if* one after another.

Technical Implementation

The database management system uses locking mechanisms and isolation levels to prevent concurrent transactions from interfering with each other.

D – Durability: Permanent Storage

Transaction Commits

All changes are successfully completed and the COMMIT command is executed.

1

Data Persisted

Changes are written to permanent storage, not just kept in temporary memory.

2

System Crash

Even if the system crashes immediately after commit, the data remains safe.

3

Recovery Complete

After restart, the database remembers that the transaction was successful and the changes persist.

4

Once a transaction is committed, the data remains safe even if the system crashes. This guarantee of permanence is what makes databases reliable for critical applications like banking and healthcare.

Serializability: Managing Concurrent Transactions

When multiple transactions run together, their actions can mix up and potentially cause incorrect results. To maintain data integrity, the Database Management System (DBMS) ensures they behave like they ran one by one.

This concept is called Serializability — executing multiple transactions so the final result is the same as some serial order.

Understanding Schedules

What is a Schedule?

A Schedule is simply an order of execution of instructions from multiple transactions. It defines the sequence in which operations from different transactions are performed.

Serial Schedule

All tasks of one transaction finish completely before the next starts.

Example: T1 completes all steps → then T2 starts and completes all steps

Non-Serial Schedule

Steps of multiple transactions are mixed (interleaved).

Example: T1 step 1 → T2 step 1 → T1 step 2 → T2 step 2

If mixing doesn't cause wrong results → it's serializable and acceptable.

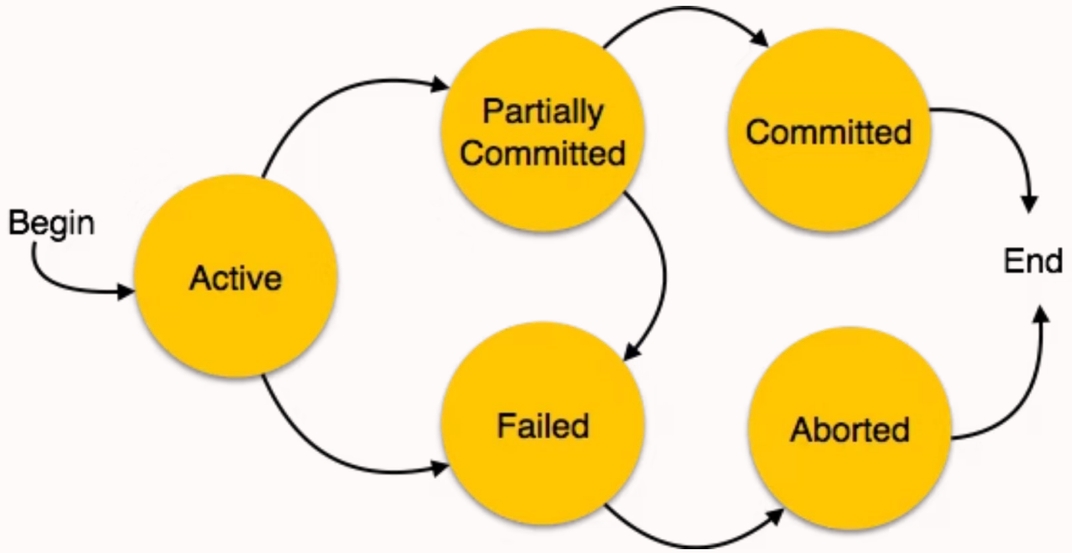
Types of Equivalence

To check if two schedules are "safe" and produce correct results, we compare them using different types of equivalence. These help determine whether a non-serial schedule is acceptable.

Type	Meaning	Example
Result Equivalence	Both schedules give the same final result	T1 + T2 produce same output values
View Equivalence	Both read/write data in same order	Same data read/written by each transaction
Conflict Equivalence	The order of conflicting operations (read/write on same data) is the same	Same write/read sequence maintained

Important Relationship: Conflict Serializable → View Serializable, but not always the other way around. Conflict serializability is a stricter condition than view serializability.

Transaction States :



The Five Transaction States

1

Active

Transaction is currently running and executing commands. This is the initial state when a transaction begins.

2

Partially Committed

All steps have been executed successfully, but changes are not yet permanently saved to the database.

3

Committed

Changes are saved permanently to the database — transaction completed successfully!

4

Failed

Some problem occurred during execution (system crash, rule violation, constraint error, etc.).

5

Aborted

Transaction is cancelled and rolled back to original state. After abort, the system can either restart or terminate the transaction.

Transaction Control Commands

COMMIT;

Purpose: Make all changes permanent and finalize the transaction.

Once committed, the changes cannot be undone and are guaranteed to persist even if the system crashes.

ROLLBACK;

Purpose: Undo all changes made during the current transaction.

Returns the database to the state it was in before the transaction began, as if the transaction never occurred.

SAVEPOINT name;

Purpose: Create a point within a transaction to roll back partially.

Allows you to undo only part of a transaction instead of rolling back everything, providing more granular control.

Consistent State Explained

Definition

A **consistent state** means all data in the database follows all the defined rules, constraints, and relationships. The database satisfies all integrity constraints and business rules.

Key Characteristics

- All constraints are satisfied
- Relationships are maintained
- Business rules are enforced
- Data integrity is preserved

Example: Money Transfer

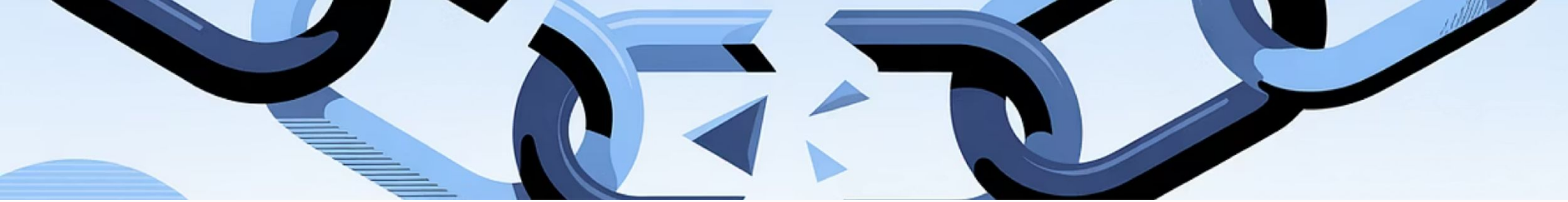
Before Transfer

A: ₹5000, B: ₹2000

After Transfer

A: ₹4000, B: ₹3000

The **total amount remains the same** (₹7000), so the database is in a consistent state before and after the transaction.



Inconsistent State: When Things Go Wrong

Definition

An inconsistent state means the data in the database is incorrect or violates constraints/rules due to an incomplete or failed transaction.

Example Scenario: Transaction Failure

- ₹1000 deducted from Account A
- System crash occurs before adding ₹1000 to Account B

The Problem

Account A: ₹4000**Account B:** ₹2000**Total:** ₹6000 ❌

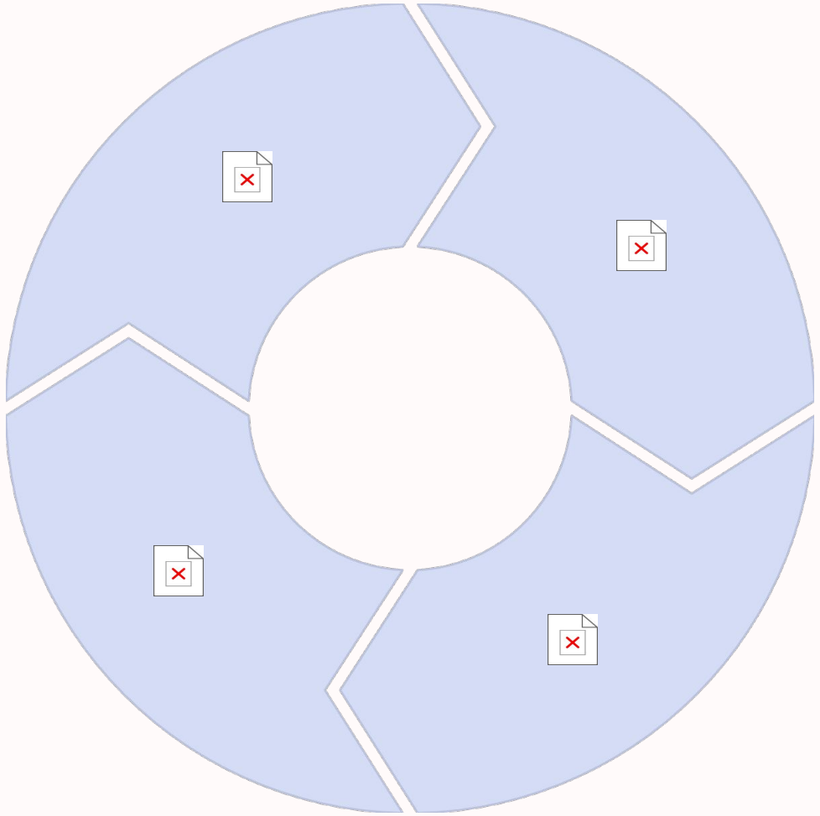
This violates the fundamental rule — **money has vanished!** The database is now in an inconsistent state.

How Databases Maintain Consistency

The **Consistency property** from ACID principles ensures that databases remain reliable and accurate through all operations.

Consistent State

Database starts in a valid state with all constraints satisfied



Transaction Executes

Operations are performed according to defined rules

New Consistent State

Transaction brings database to another valid state

Validation Check

System verifies all constraints are still satisfied

If something goes wrong during execution, the system **rolls back** to the last consistent state, ensuring data integrity is never compromised.

Consistency in Action: The Safety Net

Automatic Protection

Database management systems automatically enforce consistency through:

Constraint checking: Validates all rules before committing

Rollback mechanisms: Undoes incomplete transactions

Transaction logs: Records all changes for recovery

Validation rules: Ensures data integrity at all times

These mechanisms work together seamlessly to prevent inconsistent states from ever being permanently stored in the database.

State Comparison Summary

Term	Meaning	Example
Consistent State	All rules and constraints are satisfied. Database integrity is maintained.	Total balance before transaction = Total balance after transaction
Inconsistent State	Data violates integrity constraints. Database rules are broken.	Money lost or duplicated due to incomplete transaction

---

The fundamental goal of transaction management is to ensure the database moves from one consistent state to another, never allowing inconsistent states to persist. This is achieved through the combined enforcement of all ACID properties working together.

Thank You!