MERiT

Introduction to Transactions and Concurrency

Transactions bundle multiple database operations into a single, coherent unit of work. They are fundamental in ensuring data integrity within database management systems when multiple operations are executed simultaneously.

Concurrency control is the mechanism that ensures multiple users or processes can safely access and modify data at the same time. The key challenge is handling concurrent transactions without compromising data integrity or consistency.

In this presentation, we'll explore how databases manage transactions, the principles that govern them, and the mechanisms used to control concurrency in modern database systems.

ACID Principles

Atomicity

All operations in a transaction must complete successfully, or none do. If any operation fails, the entire transaction is rolled back, ensuring partial updates don't occur.

Consistency

Transactions must always transform the database from one valid state to another valid state, maintaining all defined rules, constraints, and triggers.

Isolation

Results of a transaction remain invisible to other transactions until it completes. This prevents intermediate states from affecting concurrent operations.

Durability

Once a transaction commits, its changes persist permanently in the database, even if system failures occur afterward.

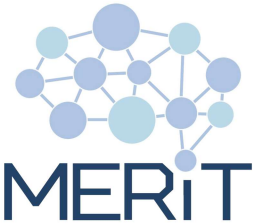
Isolation Levels

Database systems offer different isolation levels that balance performance against data consistency. Each level provides different guarantees about what concurrent transactions can observe.

READ UNCOMMITTED	Lowest isolation level. Allows reading uncommitted data, making it prone to dirty reads. Offers highest performance but lowest consistency.
READ COMMITTED	Ensures no dirty reads by only showing committed data. Each statement sees data committed before it begins execution.
REPEATABLE READ	Guarantees that rows read remain consistent within a transaction. Prevents nonrepeatable reads but may allow phantom reads.
SERIALIZABLE	Highest isolation level. Makes transactions appear to execute one after another in strict sequence. Prevents all anomalies but reduces concurrency.

Higher isolation levels provide stronger consistency guarantees but may reduce performance due to increased locking or version management overhead.

Locking and Deadlocks



Lock Types

- Shared (S) locks: Allow multiple transactions to read simultaneously
- Exclusive (X) locks: Prevent other transactions from reading or modifying
- Intent locks: Signal intention to acquire locks at a finer granularity
- Predicate locks: Lock ranges of rows matching a condition

Deadlock Scenario

Deadlocks occur when transactions lock resources in a circular dependency pattern:

1. Transaction 1 locks resource A, needs resource B
2. Transaction 2 locks resource B, needs resource A
3. Both transactions wait indefinitely for resources
4. DBMS detects and resolves by aborting one transaction

The database management system employs deadlock detection algorithms that periodically check for circular wait conditions and resolve them by selecting a victim transaction to roll back.

Dealing with Deadlocks

Prevention Strategies

Always request resources in a consistent, predetermined order to prevent circular wait conditions. This simple approach can eliminate many potential deadlock situations.

Transaction Design

Keep transactions small, focused, and short-lived. The shorter a transaction runs, the less time it holds locks and the lower the chance of conflicts with other transactions.

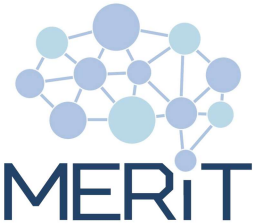
Isolation Tuning

Use the lowest isolation level that meets your consistency requirements. Lower isolation levels typically require fewer locks and reduce contention.

Retry Logic

Implement application-level deadlock handling by catching deadlock exceptions and automatically retrying transactions after a short delay, potentially with exponential backoff.

MVCC Overview



Multi-Version Concurrency Control (MVCC) is an advanced technique that creates snapshot versions of data for each transaction, allowing readers and writers to operate without blocking each other.

Version Management

Instead of locking rows, MVCC creates new versions when data is modified. Each transaction sees a consistent snapshot of data as of a specific point in time.

Read Operations

Read queries see data versions that were committed before the transaction started, ignoring newer versions. This eliminates read-write conflicts without explicit locks.

Write Operations

Write operations create new versions of data instead of overwriting existing ones. This allows concurrent reads to continue using previous versions uninterrupted.

Version Cleanup

Old versions are retained until no active transactions need them, then they're eventually removed by a garbage collection process to reclaim storage space.

MVCC in PostgreSQL and MySQL

PostgreSQL Implementation

- Uses transaction IDs (XIDs) to track versions
- Each row contains xmin (creating transaction) and xmax (deleting transaction) fields
- Vacuum process removes obsolete row versions
- Snapshot isolation is the default behavior
- Maintains a visibility map to track which pages contain obsolete tuples

MySQL (InnoDB) Implementation

- Uses rollback segments in the undo log
- Maintains a system-wide transaction list
- Row contains a transaction ID and rollback pointer
- Purge thread removes old versions when no longer needed
- REPEATABLE READ is the default isolation level

Both systems implement MVCC to achieve high concurrency while maintaining transaction isolation. The key difference lies in how they store and manage version information, with PostgreSQL using tuple headers and MySQL using undo logs.

Summary and Best Practices

1

ACID Compliance

Ensure your transactions adhere to ACID principles. Use commit and rollback operations appropriately to maintain data integrity under all circumstances.

2

Isolation Selection

Choose appropriate isolation levels based on your specific requirements. Balance consistency needs against performance implications for each transaction type.

3

Lock Management

Understand locking behavior in your DBMS. Design transactions to minimize lock duration and scope, and implement proper deadlock handling in your applications.

4

MVCC Advantages

Leverage MVCC capabilities of modern databases to achieve higher concurrency. Be aware of the version cleanup mechanisms to prevent database bloat.

The effective management of transactions and concurrency is crucial for building robust, high-performance database applications. By understanding these concepts and applying the best practices, you can ensure data integrity while maximizing system throughput.