# InnoDB Engine Deep Dive

## 3.1 ACID Properties

### What are ACID Properties?

ACID is a set of database properties that ensure reliable transactions. Only InnoDB (and similar ACID-compliant engines) guarantees these properties.

### Atomicity (A)

**Definition**: Transaction either completes fully or not at all

**Guarantee**: All statements in a transaction execute together

```
BEGIN;
UPDATE accounts SET balance = balance - 100 WHERE id = 1;
UPDATE accounts SET balance = balance + 100 WHERE id = 2;
COMMIT;  -- Both updates succeed or both fail (no partial update)
```

**Example of Atomicity Protection:**

```
-- Without atomicity (problematic)
-- If server crashes after first UPDATE, data is inconsistent

-- With ACID (InnoDB)
-- Either both succeed or both fail
BEGIN;
INSERT INTO orders (customer_id, total) VALUES (1, 500);
INSERT INTO order_items (order_id, product_id) VALUES (LAST_INSERT_ID(), 42);
COMMIT;  -- If ROLLBACK occurs, both INSERTs are undone
```

### Consistency (C)

**Definition**: Database moves from one valid state to another valid state

**Constraints Maintained:**

- Primary key uniqueness

- Foreign key relationships
- Column data types and constraints

*-- Violation attempt*

INSERT INTO users (email) VALUES (NULL);  *-- If NOT NULL constraint exists*

*-- Foreign key violation*

INSERT INTO orders (customer_id) VALUES (999);  *-- If customer 999 doesn't exist*

*-- Error: Cannot add or update a child row*

*-- Atomicity ensures consistency:*

*-- If transaction violates constraint, entire transaction rolls back*

**Isolation (I)**

**Definition**: Concurrent transactions don't interfere with each other

**Isolation Levels (increasing isolation):**

1. **READ UNCOMMITTED**

   SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

   *-- Dirty reads possible: can see uncommitted data*

   *-- Risk: Other transaction may rollback*
2. **READ COMMITTED** (Most common in production)

   SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

   *-- Only committed data visible*

   *-- Non-repeatable reads possible: data may change within transaction*
3. **REPEATABLE READ** (MySQL default)

   SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

   *-- Data read is guaranteed not to change during transaction*

   *-- Phantom reads possible: new rows may appear matching WHERE clause*

4. **SERIALIZABLE**

   SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

   *-- Highest isolation, slowest performance*

   *-- Transactions execute as if sequential*

**Isolation Levels Comparison:**

| Isolation Level | Dirty Read | Non-repeatable Read | Phantom Read |
|---|---|---|---|
| READ UNCOMMITTED | Yes | Yes | Yes |
| READ COMMITTED | No | Yes | Yes |
| REPEATABLE READ | No | No | Yes |
| SERIALIZABLE | No | No | No |

**Durability (D)**

**Definition**: Committed data persists despite failures

**Guarantees:**

- Data written to persistent storage
- Survives power failures, crashes
- Recovered on restart

*-- Once COMMIT succeeds, data is durable*

BEGIN;

INSERT INTO users (name) VALUES ('John');

COMMIT;  *-- Data is now permanent on disk*

*-- Even if server crashes immediately after COMMIT,*

*-- data survives and is recovered on restart*

**Durability Mechanisms:**

- Write-Ahead Logging (WAL): Log written before data
- Redo log: Replays committed transactions on recovery
- InnoDB recovery process: Automatic recovery on startup

**3.2 InnoDB Transactions**

**Transaction Basics**

**Starting a Transaction:**

```
BEGIN;          -- Start transaction
START TRANSACTION;  -- Alternative

-- Multiple statements within transaction
SELECT * FROM accounts WHERE id = 1 FOR UPDATE;
INSERT INTO log (action) VALUES ('Transfer started');
UPDATE accounts SET balance = balance - 100 WHERE id = 1;
UPDATE accounts SET balance = balance + 100 WHERE id = 2;
INSERT INTO transactions (from_id, to_id, amount)
VALUES (1, 2, 100);

COMMIT;  -- Save all changes
```

**COMMIT vs ROLLBACK**

**COMMIT**: Saves all changes permanently

```
BEGIN;
UPDATE inventory SET quantity = quantity - 1 WHERE product_id = 42;
SELECT quantity FROM inventory WHERE product_id = 42;  -- Shows new value
COMMIT;  -- Changes permanently saved
```

**ROLLBACK**: Discards all changes

```
BEGIN;
UPDATE inventory SET quantity = quantity - 1 WHERE product_id = 42;
ROLLBACK;  -- Changes discarded, reverts to state before BEGIN
SELECT quantity FROM inventory WHERE product_id = 42;  -- Shows original value
```

**Savepoints**

Savepoints allow partial rollback within a transaction:

```
BEGIN;
UPDATE accounts SET balance = balance - 100 WHERE id = 1;
SAVEPOINT sp1;  -- Create savepoint

UPDATE accounts SET balance = balance + 100 WHERE id = 2;
-- Error occurs!

ROLLBACK TO sp1;  -- Rollback only second update
-- First update still in transaction

UPDATE accounts SET balance = balance + 100 WHERE id = 3;
COMMIT;  -- Saves first and third updates
```

**Autocommit Mode**

By default, MySQL runs in autocommit mode:

```
-- Check autocommit status
SHOW VARIABLES LIKE 'autocommit';  -- Default: ON

-- Disable autocommit for explicit transactions
SET autocommit = 0;
-- Now each statement must be explicitly committed

-- Re-enable
SET autocommit = 1;
```

```
-- Or use explicit BEGIN to disable autocommit for one transaction
BEGIN;
-- Even with autocommit ON, transaction requires explicit COMMIT
COMMIT;
```

### 3.3 Locking and Concurrency

**Row-Level Locking**

InnoDB uses row-level locks for fine-grained concurrency control:

```
-- Shared lock (read lock)
SELECT * FROM accounts WHERE id = 1 LOCK IN SHARE MODE;

-- Exclusive lock (write lock) - used in UPDATE/DELETE
SELECT * FROM accounts WHERE id = 1 FOR UPDATE;

-- These locks prevent other transactions from modifying/reading the row
```

**Lock Conflicts Example**

**Transaction 1:**

```
BEGIN;
SELECT * FROM accounts WHERE id = 1 FOR UPDATE;  -- Acquires exclusive lock
-- Lock held until COMMIT
```

**Transaction 2 (in parallel):**

```
BEGIN;
UPDATE accounts SET balance = 500 WHERE id = 1;  -- Waits for lock
-- Blocked until Transaction 1 releases lock
```

**Lock Monitoring**

```
-- View current locks
SHOW OPEN TABLES WHERE In_use > 0;

-- View lock waits (MySQL 8.0+)
SELECT * FROM INFORMATION_SCHEMA.INNODB_LOCKS;
SELECT * FROM INFORMATION_SCHEMA.INNODB_LOCK_WAITS;
```

*-- Kill blocking transaction*

SHOW PROCESSLIST;

KILL QUERY process_id;

KILL CONNECTION process_id;

**Gap Locks and Next-Key Locks**

**Gap Lock**: Prevents insertion of new rows in a range

*-- Range query locks gap between rows*

SELECT * FROM inventory

WHERE quantity < 10

FOR UPDATE;

*-- Gap locks prevent new rows with quantity < 10 from being inserted*

**Next-Key Lock**: Combination of record and gap lock

### 3.4 InnoDB Tablespaces

**What is a Tablespace?**

Tablespace is the storage unit in InnoDB that contains table data, indexes, and metadata.

**Tablespace Types**

**1. System Tablespace (mysql)**

- Stores data dictionary
- Undo space
- Change buffer
- Configuration: innodb_data_file_path

**2. File-Per-Table Tablespaces**

[mysqld]

innodb_file_per_table = ON *# Default in MySQL 5.7+*

Each table gets its own .ibd file:

ls -lh /var/lib/mysql/database_name/

*# table1.ibd  table2.ibd  table3.ibd*

**3. General Tablespaces**

*-- Create general tablespace*

CREATE TABLESPACE ts1 ADD DATAFILE '/var/lib/mysql/ts1.ibd';

*-- Create table in specific tablespace*

CREATE TABLE inventory (

  id INT PRIMARY KEY,

  product_name VARCHAR(255)

) TABLESPACE ts1;

**4. Undo Tablespaces**

- Stores undo logs for transaction rollback

- Separate from data tablespaces
- Configuration: innodb_undo_tablespaces

**Creating and Managing Tablespaces**

```
-- List tablespaces
SELECT * FROM INFORMATION_SCHEMA.INNODB_TABLESPACES;


-- Check tablespace size
SELECT
   SPACE,
   NAME,
   FILE_TYPE,
   INITIAL_SIZE,
   CURRENT_SIZE
FROM INFORMATION_SCHEMA.INNODB_TABLESPACES
WHERE NAME = 'database_name/table_name';
```

### 3.5 Monitoring InnoDB with SHOW ENGINE INNODB STATUS

**Running the Command**

SHOW ENGINE INNODB STATUS\G

**Output Sections**

**Sections in Output:**

1. **Background Threads**: I/O threads, logger thread, master thread
2. **Semaphores**: Thread synchronization state
3. **Current Transaction State**: Active transactions and locks
4. **Pending I/O Reads/Writes**: Active I/O operations
5. **Buffer Pool and Memory**: Buffer pool usage statistics
6. **Transaction Log**: Log sequence number, checkpoint info

**Key Metrics to Monitor**

**1. Buffer Pool Efficiency**

Buffer pool hit rate = (buffer_pool_reads / buffer_pool_total_reads) * 100

Ideal: > 99%

Poor: < 98%

**2. Transaction Activity**

Number of active transactions

Transactions waiting for locks

**3. Lock Conflicts**

Number of lock waits

Deadlock detection and resolution

**Interpreting InnoDB Status**

==========

2026-01-14 16:30:45 0x7f8b3c001700

---

Trx id counter 45

Purge done for trx's n:o < 42 undo n:o < 0

History list length 0

List of transactions for each session:

---TRANSACTION 42, ACTIVE 0 sec

2 lock struct(s), heap size 1136, 0 row lock(s), undo log entries 0

MySQL thread id 8, OS thread handle 0x7f8b3c0e5700, query id 123 localhost root

init

### 3.6 Summary: Key Takeaways

1. **ACID Properties**: Ensure data reliability and consistency
    - Atomicity: All or nothing
    - Consistency: Valid state to valid state
    - Isolation: Transactions don't interfere
    - Durability: Committed data persists
2. **Transaction Control**: BEGIN, COMMIT, ROLLBACK, SAVEPOINT
3. **Isolation Levels**: REPEATABLE READ (default), READ COMMITTED recommended
4. **Locking**: Row-level locks, gap locks, deadlock avoidance
5. **Tablespaces**: File-per-table recommended, monitor with INFORMATION_SCHEMA
6. **Monitoring**: Use SHOW ENGINE INNODB STATUS for health checks