3. Transaction Logging (REDO/UNDO)

3.1 Write-Ahead Log (WAL) Structure

```
CREATE TABLE transaction_log (
    lsn BIGINT PRIMARY KEY,           -- Log Sequence Number
    transaction_id VARCHAR(10),       -- Transaction identifier
    operation_type VARCHAR(20),       -- BEGIN, UPDATE, INSERT, DELETE,
COMMIT, ROLLBACK
    table_name VARCHAR(50),           -- Affected table
    before_image TEXT,                -- Data before change
    after_image TEXT,                 -- Data after change
    timestamp TIMESTAMP DEFAULT NOW() -- When operation occurred
);
```

3.2 Sample WAL Entries

LSN	Transaction ID	Operation	Table	Before Image	After Image	Timestamp
1001	T1	BEGIN	-	-	-	10:30:15
1002	T1	UPDATE	accounts	{id:1, balance:1000}	{id:1, balance:900}	10:30:16
1003	T2	BEGIN	-	-	-	10:30:17
1004	T2	INSERT	accounts	NULL	{id:4, name:"David", balance:500}	10:30:18
1005	T1	UPDATE	accounts	{id:2, balance:500}	{id:2, balance:600}	10:30:19
1006	T1	COMMIT	-	-	-	10:30:20
1007	T2	ROLLBACK	-	-	-	10:30:21

3.3 UNDO Operations

```
-- UNDO Process for Transaction T2 (from LSN 1007 backwards)
-- LSN 1004: UNDO INSERT
DELETE FROM accounts WHERE id = 4;
```

-- Result: T2's changes are completely reversed

UNDO Algorithm:

```
def undo_transaction(transaction_id, wal_log):
    """
    Undo all operations of a transaction in reverse order
    """
    # Get all log entries for transaction in reverse order
    entries = get_log_entries(transaction_id, reverse=True)

    for entry in entries:
        if entry.operation_type == 'UPDATE':
            # Restore before image
            restore_data(entry.table_name, entry.before_image)
        elif entry.operation_type == 'INSERT':
            # Delete the inserted record
            delete_record(entry.table_name, entry.after_image)
        elif entry.operation_type == 'DELETE':
            # Re-insert the deleted record
            insert_record(entry.table_name, entry.before_image)
```

3.4 REDO Operations

```
-- REDO Process for Transaction T1 (from LSN 1002 forwards)
-- LSN 1002: REDO UPDATE
UPDATE accounts SET balance = 900 WHERE id = 1;

-- LSN 1005: REDO UPDATE
UPDATE accounts SET balance = 600 WHERE id = 2;

-- Result: T1's committed changes are reapplied
```

REDO Algorithm:

```
def redo_transaction(transaction_id, wal_log):
    """
    Redo all operations of a committed transaction in forward order
    """
    entries = get_log_entries(transaction_id, reverse=False)

    for entry in entries:
        if entry.operation_type in ['UPDATE', 'INSERT', 'DELETE']:
            # Reapply the operation using after image
            apply_operation(entry.table_name, entry.operation_type, entry.after_image)
```

3.5 Recovery Using Logs

```
-- Complete Recovery Process
-- Phase 1: Analysis
WITH uncommitted_transactions AS (
    SELECT DISTINCT transaction_id
    FROM transaction_log
    WHERE transaction_id NOT IN (
        SELECT transaction_id
        FROM transaction_log
        WHERE operation_type = 'COMMIT'
    )
),
committed_transactions AS (
    SELECT DISTINCT transaction_id
    FROM transaction_log
    WHERE transaction_id IN (
        SELECT transaction_id
        FROM transaction_log
        WHERE operation_type = 'COMMIT'
    )
)

-- Phase 2: REDO committed transactions
-- Phase 3: UNDO uncommitted transactions
```