# Hands On Lab : MySQL Client Programs & Table Maintenance

**LAB 4.1: MySQL Client Tools and Administration**

**Objective:** Master MySQL command-line tools for administration and maintenance.

**Step-by-Step Instructions:**

1. **Prepare Sample Database (Sakila)**

   ```
   # Download Sakila sample database
   cd /tmp
   wget https://dev.mysql.com/doc/index-other.html
   # Or use: mysql < sakila-schema.sql && mysql < sakila-data.sql
   ```

2. **Using mysql Command-Line Client**

   ```
   # Connect with password prompt
   mysql -u root -p -h localhost

   # Connect without password prompt (using config file)
   mysql --defaults-file=~/.my.cnf

   # Execute query directly
   mysql -u root -p -e "SELECT VERSION();"

   # Execute query from file
   mysql -u root -p < script.sql

   # Batch mode
   mysql -u root -p -B < large_script.sql > output.txt
   ```

3. **Using mysqladmin for Server Administration**

```
# Check server status
mysqladmin -u root -p status

# Create database
mysqladmin -u root -p create testdb

# Drop database
mysqladmin -u root -p drop testdb

# Flush tables
mysqladmin -u root -p flush-tables

# Flush logs
mysqladmin -u root -p flush-logs

# Show processlist
mysqladmin -u root -p processlist

# Kill process
mysqladmin -u root -p kill PROCESS_ID

# Shutdown server
mysqladmin -u root -p shutdown

# Get server variables
mysqladmin -u root -p variables | head -20
```

4. **Using mysqldump for Backups**

   ```
   # Full database backup
   mysqldump -u root -p --databases sakila > sakila_backup.sql

   # Specific table
   mysqldump -u root -p sakila actor > actor_backup.sql

   # All databases
   mysqldump -u root -p --all-databases > full_backup.sql

   # Backup with specific options
   mysqldump -u root -p \
     --single-transaction \
     --quick \
     --lock-tables=false \
     sakila > sakila_backup_consistent.sql

   # Gzip compressed backup
   mysqldump -u root -p sakila | gzip > sakila_backup.sql.gz

   # Backup with triggers and routines
   mysqldump -u root -p --triggers --routines sakila > sakila_full.sql
   ```

5. **Using mysqlpump for Advanced Backups**

   ```
   # Parallel backup (faster than mysqldump)
   mysqlpump -u root -p --parallel=4 sakila > sakila_parallel.sql
   ```

```
# Exclude specific tables
mysqlpump -u root -p sakila --exclude-tables=film,actor > sakila_partial.sql

# Backup with compression
mysqlpump -u root -p sakila --compress-output=LZ4 > sakila.lz4
```

6. **Using mysqlslap for Load Testing**

```
# Simple load test
mysqlslap -u root -p --concurrency=10 --iterations=100 \
  --auto-generate-sql

# Load test specific database
mysqlslap -u root -p --concurrency=5,10,15 \
  --auto-generate-sql --auto-generate-sql-load-type=read \
  --number-of-queries=1000 sakila

# Custom query load test
mysqlslap -u root -p --concurrency=5 \
  --iterations=10 \
  --query="SELECT * FROM sakila.actor WHERE actor_id < 100;" \
  sakila
```

**Hands-on Tasks:**

- Create, backup, and restore databases using mysqldump
- Run load tests with mysqlslap and analyze results
- Monitor active connections with mysqladmin
- Create scripts that use mysql client in batch mode

**Solution:**

1. Create, backup, and restore a test database using mysqldump (logical backup) and verify the process works end-to-end.

**Step 1: Create sample database and data**

```
-- Create test database
CREATE DATABASE test_backup_restore;

USE test_backup_restore;

-- Create sample tables
CREATE TABLE employees (
  emp_id INT PRIMARY KEY AUTO_INCREMENT,
  first_name VARCHAR(50),
  last_name VARCHAR(50),
  dept VARCHAR(30),
  salary DECIMAL(10,2),
  hire_date DATE
);

CREATE TABLE departments (
  dept_id INT PRIMARY KEY,
  dept_name VARCHAR(50),
  location VARCHAR(50)
);
```

```
-- Insert sample data
INSERT INTO departments VALUES
(1, 'Engineering', 'Hyderabad'),
(2, 'HR', 'Bangalore'),
(3, 'Finance', 'Mumbai');

INSERT INTO employees (first_name, last_name, dept, salary, hire_date) VALUES
('Alice', 'Smith', 'Engineering', 75000.00, '2024-01-15'),
('Bob', 'Johnson', 'Engineering', 82000.00, '2024-03-10'),
('Charlie', 'Brown', 'HR', 65000.00, '2024-02-20'),
('Diana', 'Davis', 'Finance', 70000.00, '2024-04-05');
```

**Step 2: Backup using mysqldump**

Execute from Linux terminal (or Windows command prompt). Assume MySQL runs on localhost:3306 with root user.

**Full database backup:**

```
mysqldump -u root -p --databases test_backup_restore > test_backup_full.sql
```

Key options explained:

- --databases: Includes CREATE DATABASE statement in backup.
- Enter password when prompted.

**Structure-only backup:**

```
mysqldump -u root -p test_backup_restore --no-data > test_backup_schema.sql
```

**Data-only backup:**

```
mysqldump -u root -p test_backup_restore --no-create-info > test_backup_data.sql
```

**Compressed backup:**

```
mysqldump -u root -p test_backup_restore | gzip > test_backup_compressed.sql.gz
```

**Verify backup file size and peek at contents:**

```
ls -lh test_backup*.sql*
head -20 test_backup_full.sql
```

**Step 3: Test restore process**

Drop and recreate target database:

```
DROP DATABASE IF EXISTS test_backup_restore_restore;
CREATE DATABASE test_backup_restore_restore;
```

Restore full backup:

```
mysql -u root -p test_backup_restore_restore < test_backup_full.sql
```

Uncompress and restore:

```
gunzip < test_backup_compressed.sql.gz | mysql -u root -p test_backup_restore_restore
```

**Step 4: Verify restore**

```
USE test_backup_restore_restore;
SHOW TABLES;
SELECT * FROM employees;
SELECT * FROM departments;
SELECT COUNT(*) FROM employees;  -- Should return 4
```

Compare original vs restored:

```
USE test_backup_restore;

SELECT COUNT(*) AS original_count FROM employees;


USE test_backup_restore_restore;

SELECT COUNT(*) AS restored_count FROM employees;
```

**Step 5: Advanced mysqldump options**

Consistent backup with transactions:

```
mysqldump -u root -p --single-transaction --routines --triggers test_backup_restore >
test_backup_consistent.sql
```

Exclude specific tables:

```
mysqldump -u root -p test_backup_restore --ignore-table=test_backup_restore.employees >
test_backup_no_employees.sql
```

Using mysqlpump (parallel alternative):

```
mysqlpump -u root -p test_backup_restore --parallel-schemas=test_backup_restore:4 >
test_pump_backup.sql
```

Expected outcomes checklist

- Backup file contains CREATE DATABASE and CREATE TABLE statements
- Backup includes all data and constraints
- Restore creates identical database structure and data
- Row counts match between original and restored tables
- Compressed backup is significantly smaller

This lab demonstrates mysqldump as a standard logical backup tool for MySQL, suitable for hot backups of InnoDB tables with --single-transaction for consistency.

2. Run load tests using mysqlslap against sample tables, then analyze the benchmark output to understand concurrency impact on MySQL performance.

**Step 1: Prepare test environment**

Use the test_backup_restore database from previous labs or create a simple test table.

```
USE test_backup_restore;


-- Ensure we have data for testing

SELECT COUNT(*) FROM employees;  -- Should have data from backup lab
```

**Step 2: Basic mysqlslap tests**

Execute from Linux terminal. Monitor MySQL slow query log or SHOW PROCESSLIST in another session to observe concurrent connections.

**Test 1: Single client, simple SELECT (baseline)**

```
mysqlslap -u root -p \
  --concurrency=1 \
  --iterations=10 \
  --query="SELECT * FROM employees" \
  --create-schema=test_backup_restore \
  --csv=basic_select.csv
```

**Test 2: High concurrency, same query**

```
mysqlslap -u root -p \
  --concurrency=50 \
  --iterations=10 \
  --query="SELECT * FROM employees WHERE dept='Engineering'" \
  --create-schema=test_backup_restore \
  --csv=concurrency_50.csv
```

**Test 3: Mixed read/write workload**

```
mysqlslap -u root -p \
  --concurrency=20 \
  --iterations=5 \
  --query="INSERT INTO employees (first_name, last_name, dept, salary, hire_date) VALUES
('Test', 'User', 'QA', 60000.00, NOW()); SELECT * FROM employees" \
  --create-schema=test_backup_restore \
  --csv=read_write.csv
```

**Auto-generated load test (no query needed):**

```
mysqlslap -u root -p \
  --concurrency=100 \
  --iterations=5 \
  --auto-generate-sql \
  --auto-generate-sql-load-type=mixed \
  --auto-generate-sql-execute-number=20 \
  --create-schema=test_backup_restore \
  --csv=auto_generated.csv
```

**Step 3: Sample output analysis**

Typical mysqlslap output shows these key metrics:

```
Benchmark
    Average number of seconds to run all queries: 0.197 seconds
    Minimum number of seconds to run all queries: 0.168 seconds
    Maximum number of seconds to run all queries: 0.399 seconds
    Number of clients running queries: 50
    Average number of queries per client: 1
```

CSV output (from --csv flag) for Excel analysis:

```
"iterations","total_time","avg_secs","min_secs","max_secs","concurrency","avg_queries"
"10","1.97","0.197","0.168","0.399","50","1"
```

**Step 4: Analyze and compare results**

Create comparison table from CSV outputs:

| Test Type | Concurrency | Iterations | Avg Time (s) | Min (s) | Max (s) | QPS (queries/sec) |
|---|---|---|---|---|---|---|
| Basic SELECT | 1 | 10 | 0.015 | 0.012 | 0.020 | 666 |
| Concurrency 50 | 50 | 10 | 0.197 | 0.168 | 0.399 | 51 |
| Read/Write | 20 | 5 | 2.450 | 2.100 | 2.890 | 4 |
| Auto-generated | 100 | 5 | 15.320 | 12.500 | 18.200 | 7 |

**Calculate QPS (Queries Per Second):**

QPS = (concurrency × iterations) / average_seconds

**Key observations to discuss:**

- As concurrency increases, average time per iteration rises due to resource contention.
- High variance (min/max difference) indicates inconsistent performance under load.
- Write-heavy workloads (INSERT+SELECT) are significantly slower than reads.
- Auto-generated mixed load simulates real application patterns best.

**Step 5: Performance monitoring during tests**

In another terminal, watch MySQL performance:

```
-- Monitor active connections
SHOW PROCESSLIST;
```

```
-- Check InnoDB status
SHOW ENGINE INNODB STATUS\G

-- Global status before/after
SHOW GLOBAL STATUS LIKE 'Queries';
SHOW GLOBAL STATUS LIKE 'Connections';
```

Expected improvements after tuning:

- Increase innodb_buffer_pool_size
- Adjust table_open_cache
- Add indexes on dept column: CREATE INDEX idx_dept ON employees(dept);

**Lab verification checklist**

- mysqlslap completes without connection errors
- CSV files generated with benchmark metrics
- Higher concurrency shows degraded performance
- QPS calculation matches expected degradation pattern
- MySQL processlist shows multiple concurrent threads during test

This lab demonstrates mysqlslap as a quick load generator for capacity planning and tuning validation, with output metrics directly comparable across MySQL configurations.

3. Monitor active MySQL connections using mysqladmin processlist and related commands to observe connection states during workload.

**Step 1: Basic connection monitoring**

Run these from Linux terminal while other sessions (or mysqlslap from previous lab) generate load.

List active connections:

mysqladmin -u root -p processlist

Extended process list (full query text):

mysqladmin -u root -p processlist -v

The -v flag shows the full SQL query in the Info column instead of truncated version.

Sample output during load test:

```
+-----+-------+------------+---------+--------------+--------+-------------+----------------------------+
| Id  | User| Host       | db     | Command| Time| State       | Info                       |
+-----+------+--------------+--------+---------------+------+-------------+----------------------------+
|  5  | root | localhost |          | Query       |    0  | init          | SHOW PROCESSLIST   |
| 10 | root | localhost | sakila | Query       |  12 | executing | SELECT * FROM film   |
| 11 | root | localhost | sakila | Query       |    8 | executing | SELECT * FROM actor |
+-----+-------+------------+---------+-------------+------+-------------+----------------------------+
```

**Step 2: Continuous monitoring**

Monitor connections every 2 seconds:

mysqladmin -u root -p -i 2 -c 20 processlist

- -i 2: Interval of 2 seconds
- -c 20: Run 20 iterations (40 seconds total)

Combine processlist with status:

```
mysqladmin -u root -p processlist extended-status
```

Key status counters to watch during load:

```
Uptime: 12345
Threads: 25         # Total thread count
Questions: 15432    # Total queries executed
Connections: 156    # Total connection attempts
```

**Step 3: Generate load to observe**

Terminal 1: Start mysqlslap load test (from previous lab):

```
mysqlslap -u root -p --concurrency=20 --iterations=30 \
  --query="SELECT * FROM employees" \
  --create-schema=test_backup_restore
```

Terminal 2: Monitor active connections:

```
watch -n 1 'mysqladmin -u root -p processlist | grep -E "(Query|Execute)"'
```

Expected observation: 20+ "Query" state threads from mysqlslap clients.

**Step 4: Kill problematic connections**

Kill a specific connection (use ID from processlist):

```
mysqladmin -u root -p kill 123
```

Kill all connections from specific host:

```
mysqladmin -u root -p processlist | grep '192.168.1.100' | awk '{print $2}' | xargs -I {} mysqladmin -u root -p kill {}
```

**Step 5: Health checks**

Server ping (connection test):

```
mysqladmin -u root -p ping
```

Returns mysqld is alive if server responds.

Server status snapshot:

```
mysqladmin -u root -p status
```

```
Uptime: 12345  Threads: 3  Questions: 15432  Slow queries: 0  Opens: 85  Flush tables: 1
Open tables: 78  Queries per second avg: 1.250
```

**Step 6: Analysis checklist**

During/after load test, verify:

| Metric | Expected Observation | Action if Abnormal |
|---|---|---|
| Threads count | Matches --concurrency parameter | Check max_connections |
| Command=Query | Multiple active queries | Normal under load |
| Time > 10s | Long-running queries | Investigate slow query log |
| State=Locked | Table/row locks | Check indexes, deadlocks |
| Info column | Actual SQL being executed | Identify problematic patterns |

Common troubleshooting:

```
# If too many connections error occurs
mysqladmin -u root -p variables | grep max_connections
```

```
# Check slow query log
mysqladmin -u root -p variables | grep slow_query_log
```

This hands-on demonstrates mysqladmin processlist as a quick external monitoring tool for connection diagnostics without entering MySQL shell.

4. Create practical shell scripts that execute multiple MySQL commands in batch mode using input redirection and the mysql client.

**Step 1: Create sample SQL script files**

File: backup_check.sql

```
-- Batch script: Check backup status and table counts
USE test_backup_restore;
SELECT 'employees table' AS table_name, COUNT(*) AS row_count FROM employees;
SELECT 'departments table' AS table_name, COUNT(*) AS row_count FROM departments;
SHOW TABLE STATUS LIKE 'employees';
SHOW VARIABLES LIKE 'innodb_buffer_pool_size';
```

File: user_audit.sql

```
-- Batch script: User and privilege audit
SELECT User, Host FROM mysql.user;
SELECT 'Current user privileges' AS info;
SHOW GRANTS FOR CURRENT_USER();
SELECT CONCAT('Connections: ', variable_value) AS status
FROM information_schema.global_status
WHERE variable_name = 'Threads_connected';
```

**Step 2: Execute scripts in batch mode**

Basic batch execution:

```
mysql -u root -p < backup_check.sql
```

With verbose output (shows SQL being executed):

```
mysql -u root -p -v < backup_check.sql
```

Tabular format in batch mode:

```
mysql -u root -p -t < backup_check.sql > backup_report.txt
```

Specify database and suppress column headers:

```
mysql -u root -p test_backup_restore -N < backup_check.sql > clean_output.txt
```

**Step 3: Create automation shell scripts**

File: daily_health_check.sh

```
#!/bin/bash
# MySQL Daily Health Check Script
TIMESTAMP=$(date +%Y%m%d_%H%M%S)
LOGFILE="/tmp/mysql_health_${TIMESTAMP}.txt"

echo "MySQL Health Check - $(date)" | tee -a $LOGFILE

mysql -u root -p${MYSQL_ROOT_PASS} -e "SELECT VERSION();" | tee -a $LOGFILE

mysql -u root -p${MYSQL_ROOT_PASS} -t << EOF | tee -a $LOGFILE
USE test_backup_restore;
```

```
SELECT
  ROUND((SELECT variable_value FROM information_schema.global_status WHERE variable_name='Uptime')/3600, 2) AS uptime_hours,
  variable_value AS threads_connected
FROM information_schema.global_status
WHERE variable_name='Threads_connected';
SHOW STATUS LIKE 'Queries%';
EOF

echo "Health check complete. Log: $LOGFILE"
```

Make executable and run:

```
chmod +x daily_health_check.sh
echo "export MYSQL_ROOT_PASS='yourpassword'" >> ~/.bashrc
source ~/.bashrc
./daily_health_check.sh
```

**Step 4: Advanced batch script with error handling**

File: deploy_schema.sql

```
-- Schema deployment script with validation
SET @deploy_start = NOW();

-- Create database if not exists
CREATE DATABASE IF NOT EXISTS production_app;

USE production_app;
```

```
-- Create tables
CREATE TABLE IF NOT EXISTS orders (
  order_id INT PRIMARY KEY AUTO_INCREMENT,
  customer_id INT,
  amount DECIMAL(10,2),
  order_date DATETIME DEFAULT CURRENT_TIMESTAMP
);

-- Insert test data
INSERT IGNORE INTO orders (customer_id, amount) VALUES
(1, 99.99), (2, 149.50), (3, 75.25);

-- Validation queries
SET @deploy_end = NOW();
SELECT 'Deployment Status' AS status,
       COUNT(*) AS orders_created
FROM orders;
SELECT CONCAT('Deploy time: ', TIMEDIFF(@deploy_end, @deploy_start)) AS
deployment_time;
```

Deploy with validation:

```
mysql -u root -p -v -t < deploy_schema.sql > deploy_$(date +%Y%m%d).log
grep "Deployment Status" deploy_*.log
```

**Step 5: Conditional batch execution**

File: smart_backup.sh

```
#!/bin/bash

DB_NAME="test_backup_restore"

BACKUP_DIR="/backup"

DATE=$(date +%Y%m%d_%H%M)


# Check if backup needed (less than 1 day old)

if [ -f "${BACKUP_DIR}/${DB_NAME}_${DATE}.sql.gz" ]; then

   echo "Recent backup exists, skipping..."

   exit 0

fi


# Batch backup with row count verification

mysql -u root -p -N -e "USE ${DB_NAME}; SELECT COUNT(*) FROM employees;" >
/tmp/rowcount_before.txt

ROWCOUNT=$(cat /tmp/rowcount_before.txt)


mysqldump -u root -p ${DB_NAME} | gzip > ${BACKUP_DIR}/${DB_NAME}_${DATE}.sql.gz


mysql -u root -p -N -e "USE ${DB_NAME}; SELECT COUNT(*) FROM employees;" >
/tmp/rowcount_after.txt

NEW_ROWCOUNT=$(cat /tmp/rowcount_after.txt)


if [ "$ROWCOUNT" = "$NEW_ROWCOUNT" ]; then

   echo "SUCCESS: Backup completed. Rows unchanged: $ROWCOUNT"

else

   echo "WARNING: Row count changed during backup!"

fi
```

**Expected Results**

batch_check.sql output:

```
table_name          row_count
employees table       4
departments table     3
```

Key batch mode flags:

- -t: Tabular output format
- -v: Verbose (echo SQL commands)
- -N: No column headers
- -s: Silent mode
- -e "sql": Execute single command

**Lab verification checklist**

- .sql files execute without interactive prompts
- -t produces readable table format
- Shell script runs end-to-end with logging
- Error handling prevents partial deployments
- Output files contain expected results

This lab demonstrates batch mode for automation, scheduled tasks, and deployment scripts - essential for production DBA workflows.

**LAB 4.2: Table Maintenance Operations**

**Objective:** Perform table maintenance tasks to optimize performance and integrity.

**Step-by-Step Instructions:**

1. **Create Test Tables with Data**

```
CREATE DATABASE maintenance_test;
USE maintenance_test;

CREATE TABLE test_innodb (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(255),
  email VARCHAR(255),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  INDEX idx_name (name),
  INDEX idx_email (email)
) ENGINE=InnoDB;

-- Insert sample data
INSERT INTO test_innodb (name, email) VALUES
  ('John Doe', 'john@example.com'),
  ('Jane Smith', 'jane@example.com'),
  ('Bob Johnson', 'bob@example.com');
```

2. **CHECK Table Operation**

```
-- Check table for errors
CHECK TABLE test_innodb;

-- Extended check
CHECK TABLE test_innodb EXTENDED;
```

```
-- Check multiple tables
CHECK TABLE test_innodb, another_table;
```

3. **ANALYZE Table Operation**

```
-- Analyze table statistics
ANALYZE TABLE test_innodb;

-- View table statistics
SELECT
   TABLE_NAME,
   TABLE_ROWS,
   AVG_ROW_LENGTH,
   DATA_LENGTH,
   INDEX_LENGTH
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'maintenance_test'\G
```

4. **OPTIMIZE Table Operation**

```
-- Reclaim unused space
OPTIMIZE TABLE test_innodb;

-- View space before and after
SELECT
   TABLE_NAME,
   ROUND(((DATA_LENGTH + INDEX_LENGTH) / 1024 / 1024), 2) AS size_mb
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'maintenance_test';
```

5. **REPAIR Table Operation (MyISAM)**

```
-- Create MyISAM table for testing
CREATE TABLE test_myisam (
   id INT PRIMARY KEY AUTO_INCREMENT,
   data VARCHAR(255)
) ENGINE=MyISAM;

-- Repair table
REPAIR TABLE test_myisam;
REPAIR TABLE test_myisam QUICK;
REPAIR TABLE test_myisam EXTENDED;
```

6. **Test Table Fragmentation**

```
-- Insert and delete to fragment
INSERT INTO test_innodb (name, email) SELECT name, email FROM test_innodb;
INSERT INTO test_innodb (name, email) SELECT name, email FROM test_innodb;
INSERT INTO test_innodb (name, email) SELECT name, email FROM test_innodb;

-- Delete random rows
DELETE FROM test_innodb WHERE id % 3 = 0;

-- Check size before optimization
SELECT
   TABLE_NAME,
   ROUND(((DATA_LENGTH + INDEX_LENGTH) / 1024 / 1024), 2) AS size_before
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_NAME = 'test_innodb';
```

```
OPTIMIZE TABLE test_innodb;


-- Check size after optimization
SELECT
  TABLE_NAME,
  ROUND(((DATA_LENGTH + INDEX_LENGTH) / 1024 / 1024), 2) AS size_after
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_NAME = 'test_innodb';
```

7. **Monitor InnoDB Background Operations**

```
SHOW ENGINE INNODB STATUS\G
-- Look for: "Background thread"
```

**Hands-on Tasks:**

- Schedule OPTIMIZE TABLE for regular maintenance
- Set up automated CHECK TABLE jobs
- Monitor table fragmentation over time
- Create maintenance scripts combining multiple operations

**Solution:**

1. Schedule regular table maintenance using mysqlcheck and cron jobs, or MySQL Event Scheduler for automated OPTIMIZE TABLE operations on fragmented tables.

**Step 1: Create maintenance SQL script**

**File: /opt/mysql_maintenance/optimize_tables.sql**

```
-- Weekly table optimization script
USE test_backup_restore;
```

```
SELECT 'OPTIMIZING TABLES - START' AS status, NOW() AS timestamp;


OPTIMIZE TABLE employees;

OPTIMIZE TABLE departments;


SELECT 'OPTIMIZATION COMPLETE' AS status, NOW() AS timestamp;


-- Report fragmentation status

SELECT

  table_name,

  engine,

  ROUND(data_free / 1024 / 1024, 2) AS data_free_mb,

  table_rows,

  data_length / 1024 / 1024 AS data_mb

FROM information_schema.tables

WHERE table_schema = 'test_backup_restore';
```

**Step 2: Create bash wrapper script**

**File: /opt/mysql_maintenance/run_optimize.sh**

```
#!/bin/bash

LOG_DIR="/var/log/mysql_maintenance"

DATE=$(date +%Y%m%d_%H%M%S)


mkdir -p $LOG_DIR


# Run optimization

mysql -u root -p${MYSQL_ROOT_PASS} < /opt/mysql_maintenance/optimize_tables.sql >
${LOG_DIR}/optimize_${DATE}.log 2>&1
```

```
# Email results if space reclaimed > 10MB
RECLAIMED=$(grep -A 10 "OPTIMIZING TABLES" ${LOG_DIR}/optimize_${DATE}.log | grep -oP '\d+\.\d+.*MB' | head -1 | cut -d' ' -f1)

if (( $(echo "$RECLAIMED > 10" | bc -l) )); then
    echo "Maintenance completed. Reclaimed: ${RECLAIMED}MB" | mail -s "MySQL Maintenance Alert" admin@company.com
fi
```

```
chmod +x /opt/mysql_maintenance/run_optimize.sh
```

**Step 3: Method 1 - Cron job scheduling**

**Edit crontab:**

```
crontab -e
```

**Weekly Sunday 2AM maintenance:**

```
0 2 * * 0 /opt/mysql_maintenance/run_optimize.sh
```

**Daily lightweight check (mysqlcheck):**

```
59 23 * * * mysqlcheck -o --all-databases -u root -p${MYSQL_ROOT_PASS} >> /var/log/mysqlcheck.log 2>&1
```

**Options explained:**

- -o: Optimize tables
- --all-databases: All DBs (exclude system DBs in production)

**Step 4: Method 2 - MySQL Event Scheduler**

```
-- Enable event scheduler
SET GLOBAL event_scheduler = ON;
SHOW VARIABLES LIKE 'event_scheduler';
```

**Create weekly optimization event:**

```
DELIMITER $$

CREATE EVENT IF NOT EXISTS weekly_table_optimize
ON SCHEDULE EVERY 1 WEEK
STARTS '2026-01-26 02:00:00'
DO
BEGIN
  OPTIMIZE TABLE test_backup_restore.employees;
  OPTIMIZE TABLE test_backup_restore.departments;
END$$

DELIMITER ;

-- View scheduled events
SHOW EVENTS FROM test_backup_restore;
```

**Step 5: Verify and monitor**

**Check cron execution:**

```
tail -f /var/log/mysql_maintenance/optimize_*.log
```

**Check Event Scheduler:**

```
SELECT event_name, status, last_executed
FROM information_schema.events
WHERE event_schema = 'test_backup_restore';
```

**Before/after fragmentation comparison:**

```
-- Run before maintenance
SELECT table_name, ROUND(data_free/1024/1024,2) as fragmented_mb
FROM information_schema.tables
WHERE table_schema='test_backup_restore';

-- Trigger manual optimization
OPTIMIZE TABLE test_backup_restore.employees;

-- Run after
SELECT table_name, ROUND(data_free/1024/1024,2) as fragmented_mb
FROM information_schema.tables
WHERE table_schema='test_backup_restore';
```

**Step 6: Production considerations**

**Smart optimization (only fragmented tables):**

```
-- Dynamic optimization script
SELECT CONCAT('OPTIMIZE TABLE ', table_schema, '.', table_name, ';') AS optimize_cmd
INTO OUTFILE '/tmp/fragmented_tables.sql'
FROM information_schema.tables
WHERE table_schema NOT IN ('mysql','information_schema','performance_schema','sys')
  AND engine IN ('MyISAM','InnoDB')
  AND data_free > 10485760;  -- >10MB fragmented
```

**Cron entry for smart optimization:**

```
0 1 * * 0 mysql -u root -pPASS < /tmp/fragmented_tables.sql
```

**Expected Results Table**

| **Method** | **Schedule** | **Scope** | **Best For** |
|---|---|---|---|
| mysqlcheck -o | Daily/Weekly | All tables | Simple automation |
| OPTIMIZE TABLE | Weekly | Specific tables | Targeted maintenance |
| Event Scheduler | Weekly | Schema-specific | MySQL-native |
| Dynamic script | Monthly | Fragmented only | Production efficiency |

**Verification Checklist**

- Cron job runs without password prompts (use ~/.my.cnf)
- Event Scheduler shows ENABLED status
- data_free decreases after optimization
- Log files capture before/after metrics
- No locks during peak hours

**Pro tip:** For InnoDB, OPTIMIZE TABLE = ALTER TABLE ... ENGINE=InnoDB (rebuilds table). Use during low-traffic windows.

2. Schedule automated CHECK TABLE operations using mysqlcheck with cron jobs to proactively detect and repair MySQL table corruption.

**Step 1: Create CHECK TABLE maintenance script**

**File: /opt/mysql_maintenance/check_tables.sql**

```
-- Daily CHECK TABLE validation script
USE test_backup_restore;

SELECT 'CHECK TABLE - START' AS status, NOW() AS timestamp;

-- Check all tables (reports OK/ERROR/Warning)
CHECK TABLE employees;
CHECK TABLE departments;

SELECT 'CHECK TABLE - COMPLETE' AS status, NOW() AS timestamp;

-- Summary report from Information Schema
SELECT
  table_name,
  table_status,
  engine,
  check_time,
  CASE
    WHEN table_status = 'OK' THEN 'PASS'
    ELSE 'FAILURE'
  END AS status
FROM information_schema.tables
```

```
WHERE table_schema = 'test_backup_restore';
```

**Step 2: Create comprehensive bash maintenance script**

**File: /opt/mysql_maintenance/check_and_repair.sh**

```
#!/bin/bash
LOG_DIR="/var/log/mysql_maintenance"
DATE=$(date +%Y%m%d_%H%M%S)
DB_NAME="test_backup_restore"

mkdir -p $LOG_DIR

echo "=== MySQL CHECK TABLE - $(date) ===" > ${LOG_DIR}/check_${DATE}.log

# Run CHECK TABLE
mysql -u root -p${MYSQL_ROOT_PASS} test_backup_restore -e "
CHECK TABLE employees, departments;
SELECT 'Check complete' AS status;
" >> ${LOG_DIR}/check_${DATE}.log 2>&1

# Check for repair needs and auto-repair
mysqlcheck -u root -p${MYSQL_ROOT_PASS} --check --auto-repair --databases $DB_NAME
>> ${LOG_DIR}/check_${DATE}.log 2>&1

# Email if errors found
if grep -qi "error\|corrupt\|failed" ${LOG_DIR}/check_${DATE}.log; then
   cat ${LOG_DIR}/check_${DATE}.log | mail -s "MySQL Table Check FAILURE - $DB_NAME"
admin@company.com
   echo "URGENT: Table corruption detected!" | tee -a ${LOG_DIR}/check_${DATE}.log
```

```
else
    echo "All tables OK - $(date)" >> ${LOG_DIR}/check_${DATE}.log
fi
```

```
chmod +x /opt/mysql_maintenance/check_and_repair.sh
```

**Step 3: Configure cron schedules**

**Edit crontab for automated execution:**

```
crontab -e
```

**Multi-level checking strategy:**

```
# Daily lightweight check (1AM)
0 1 * * * /opt/mysql_maintenance/check_and_repair.sh

# Weekly comprehensive check (Sunday 3AM)
0 3 * * 0 mysqlcheck -c --all-databases -u root -p${MYSQL_ROOT_PASS} >>
/var/log/mysqlcheck_weekly.log 2>&1

# Monthly repair (1st day 4AM)
0 4 1 * * mysqlcheck -r --all-databases -u root -p${MYSQL_ROOT_PASS} >>
/var/log/mysqlcheck_monthly.log 2>&1
```

**mysqlcheck flags explained:**

- -c, --check: Check tables for errors
- -r, --repair: Auto-repair corrupted tables
- -o, --optimize: Optimize after repair
- --auto-repair: Repair without confirmation

**Step 4: Secure credentials with MySQL config file**

**File: ~/.my.cnf (root user)**

```
[client]

user=root

password=your_secure_password
```

**Set secure permissions:**

```
chmod 600 ~/.my.cnf
```

Now cron jobs run without password prompts:

```
mysqlcheck -c --databases test_backup_restore
```

**Step 5: Test and verify automation**

**Manual test:**

```
/opt/mysql_maintenance/check_and_repair.sh

tail -20 /var/log/mysql_maintenance/check_*.log
```

**Simulate table corruption (MyISAM only):**

```
ALTER TABLE departments ENGINE=MyISAM;  -- Convert to MyISAM for testing

-- Corrupt test (use with caution in lab only)

myisamchk -w /var/lib/mysql/test_backup_restore/departments.MYI
```

**Expected CHECK TABLE output:**

```
employees: OK

departments: OK          # Normal result

departments: error: Table is crashed and last repair failed
```

```
# Triggered repair needed
```

**Step 6: Monitoring dashboard query**

**Create monitoring view:**

```
CREATE OR REPLACE VIEW maintenance_status AS
SELECT
  table_schema AS database_name,
  table_name,
  engine,
  check_time,
  checksum,
  create_time,
  update_time,
  CASE
    WHEN check_time IS NULL THEN 'Never checked'
    WHEN check_time > DATE_SUB(NOW(), INTERVAL 1 DAY) THEN 'Recent OK'
    ELSE 'Old check'
  END AS check_status
FROM information_schema.tables
WHERE table_schema = 'test_backup_restore';
```

**Expected Results Summary**

| Status | CHECK TABLE Result | Action Required |
|---|---|---|
| OK | Table passed | None |
| Warning | Minor issues | Monitor |
| Error/Corrupt | Corruption detected | Auto-repair |

| Status | CHECK TABLE Result | Action Required |
|---|---|---|
| Repaired | Fixed automatically | Verify data |

**Production Checklist**

- ~/.my.cnf permissions 600 (root only)
- Log rotation for /var/log/mysql_maintenance/*
- Exclude system databases (mysql, performance_schema)
- Alert only on actual failures (not informational warnings)
- Test repair on MyISAM tables (InnoDB self-repairs)

This creates enterprise-grade automated table integrity monitoring that detects corruption before it impacts applications.

3. Monitor MySQL table fragmentation trends over time using automated collection from INFORMATION_SCHEMA and create historical analysis reports.

**Step 1: Create fragmentation monitoring table**

```
USE test_backup_restore;

CREATE TABLE IF NOT EXISTS table_fragmentation_history (
  id INT PRIMARY KEY AUTO_INCREMENT,
  check_timestamp DATETIME DEFAULT CURRENT_TIMESTAMP,
  table_name VARCHAR(64),
  database_name VARCHAR(64),
  engine VARCHAR(64),
  table_rows BIGINT,
  data_length BIGINT,
  index_length BIGINT,
  data_free BIGINT,
  fragmentation_pct DECIMAL(5,2),
  total_size_mb DECIMAL(10,2),
  INDEX idx_timestamp (check_timestamp),
  INDEX idx_table (database_name, table_name)
);
```

**Step 2: Fragmentation monitoring script**

**File: /opt/mysql_maintenance/monitor_fragmentation.sh**

```
#!/bin/bash
LOG_DIR="/var/log/mysql_maintenance"
```

```
DATE=$(date +%Y%m%d_%H%M%S)


# Collect fragmentation data
mysql -u root -p${MYSQL_ROOT_PASS} -D test_backup_restore -N -e "
INSERT INTO table_fragmentation_history (
  table_name, database_name, engine, table_rows,
  data_length, index_length, data_free,
  fragmentation_pct, total_size_mb
)
SELECT
  table_name, 'test_backup_restore' AS database_name, engine,
  table_rows, data_length, index_length, data_free,
  ROUND((data_free / (data_length + index_length + 1)) * 100, 2) AS fragmentation_pct,
  ROUND((data_length + index_length + data_free) / 1024 / 1024, 2)
FROM information_schema.tables
WHERE table_schema = 'test_backup_restore'
  AND table_name IN ('employees', 'departments');
" >> ${LOG_DIR}/fragmentation_${DATE}.log 2>&1


echo "Fragmentation snapshot captured: $(date)" >>
${LOG_DIR}/fragmentation_${DATE}.log
```

**Step 3: Schedule frequent monitoring**

**Cron job - Every 6 hours:**

```
crontab -e
0 */6 * * * /opt/mysql_maintenance/monitor_fragmentation.sh
```

**Step 4: Generate fragmentation analysis reports**

**Current fragmentation status:**

```
SELECT
  table_name,
  engine,
  table_rows,
  ROUND(data_free/1024/1024, 2) AS fragmented_mb,
  ROUND((data_free/(data_length+index_length+1))*100, 2) AS frag_pct,
  CASE
    WHEN data_free > 10485760 THEN 'HIGH (>10MB)'
    WHEN data_free > 1048576 THEN 'MEDIUM (>1MB)'
    ELSE 'OK'
  END AS action_needed
FROM information_schema.tables
WHERE table_schema = 'test_backup_restore';
```

**30-day fragmentation trend:**

```
SELECT
  DATE(check_timestamp) AS check_date,
  table_name,
  AVG(fragmentation_pct) AS avg_frag_pct,
  AVG(data_free/1024/1024) AS avg_frag_mb,
  COUNT(*) AS samples
FROM table_fragmentation_history
WHERE check_timestamp > NOW() - INTERVAL 30 DAY
GROUP BY DATE(check_timestamp), table_name
ORDER BY table_name, check_date;
```

**Fragmentation growth rate (alert query):**

```
SELECT
  table_name,
  ROUND(MAX(fragmentation_pct) - MIN(fragmentation_pct), 2) AS pct_growth_30d,
  ROUND(MAX(data_free) - MIN(data_free), 0)/1024/1024 AS mb_growth_30d,
  MIN(check_timestamp) AS first_check,
  MAX(check_timestamp) AS last_check
FROM table_fragmentation_history
WHERE check_timestamp > NOW() - INTERVAL 30 DAY
GROUP BY table_name
HAVING pct_growth_30d > 5.0;
```

**Step 5: Simulate fragmentation for testing**

**Create fragmentation (DELETE + INSERT to fragment):**

```
-- Delete 70% of rows to create fragmentation
DELETE FROM employees WHERE emp_id > 2;

-- Insert new rows (creates fragmentation)
INSERT INTO employees (first_name, last_name, dept, salary, hire_date) VALUES
('Test1', 'User1', 'QA', 55000, NOW()),
('Test2', 'User2', 'DevOps', 65000, NOW()),
('Test3', 'User3', 'QA', 58000, NOW());
```

**Verify increased fragmentation:**

```
SHOW TABLE STATUS FROM test_backup_restore LIKE 'employees'\G
-- Note increasing Data_free value
```

**Step 6: Automated alerting script**

**File: alert_fragmentation.sh**

```
#!/bin/bash
ALERT_THRESHOLD=20  # 20% fragmentation

HIGH_FRAG=$(mysql -u root -p${MYSQL_ROOT_PASS} -N -e "
SELECT COUNT(*)
FROM information_schema.tables
WHERE table_schema='test_backup_restore'
  AND (data_free/(data_length+index_length+1))*100 > $ALERT_THRESHOLD;
")

if [ "$HIGH_FRAG" -gt 0 ]; then
    mysql -u root -p${MYSQL_ROOT_PASS} -N -e "
    SELECT table_name, ROUND((data_free/1024/1024),2) AS frag_mb
    FROM information_schema.tables
    WHERE table_schema='test_backup_restore'
      AND (data_free/(data_length+index_length+1))*100 > $ALERT_THRESHOLD;" |
    mail -s "ALERT: High table fragmentation detected" dba-team@company.com
fi
```

**Daily alert check:**

```
0 22 * * * /opt/mysql_maintenance/alert_fragmentation.sh
```

**Expected Output Example**

**Fragmentation trend table (after 1 week):**

| Date | Table | Frag % | Frag MB | Status |
|---|---|---|---|---|
| 2026-01-21 | employees | 2.1 | 0.15 | OK |
| 2026-01-22 | employees | 8.7 | 0.89 | Medium |
| 2026-01-23 | employees | 24.3 | 3.42 | HIGH |

**Lab Verification Checklist**

- table_fragmentation_history populates every 6 hours
- Fragmentation % increases after DELETE+INSERT
- Trend query shows growth over time
- Alert triggers at 20% threshold
- data_free correlates with fragmentation %

**Key Insight:** Fragmentation >15-20% typically warrants OPTIMIZE TABLE during maintenance windows.

4. Create comprehensive MySQL maintenance scripts that combine CHECK, ANALYZE, OPTIMIZE, and fragmentation monitoring in a single automated workflow.

**Step 1: Master maintenance script**

**File: /opt/mysql_maintenance/comprehensive_maintenance.sql**

```
-- ===================================================
-- MySQL Comprehensive Maintenance Script
-- Combines: CHECK + ANALYZE + OPTIMIZE + Fragmentation
-- ===================================================

SET @maintenance_start = NOW();
SET SESSION sql_log_bin = 0;  -- Disable binary logging for maintenance

SELECT '=== MAINTENANCE START ===' AS status, @maintenance_start AS timestamp;

-- 1. TABLE INTEGRITY CHECK
SELECT '1. CHECK TABLE Results:' AS phase;
CHECK TABLE test_backup_restore.employees EXTENDED;
CHECK TABLE test_backup_restore.departments EXTENDED;

-- 2. ANALYZE TABLE STATISTICS (update optimizer stats)
SELECT '2. ANALYZE TABLE Results:' AS phase;
ANALYZE TABLE test_backup_restore.employees;
ANALYZE TABLE test_backup_restore.departments;

-- 3. FRAGMENTATION ANALYSIS
SELECT '3. Fragmentation Analysis:' AS phase;
```

```
SELECT
  table_name,
  engine,
  ROUND(table_rows/1000,1) AS rows_k,
  ROUND(data_free/1024/1024,2) AS frag_mb,
  ROUND((data_free/(data_length+index_length+1))*100,1) AS frag_pct,
  CASE
    WHEN data_free > 50*1024*1024 THEN 'CRITICAL'
    WHEN data_free > 10*1024*1024 THEN 'HIGH'
    WHEN data_free > 1*1024*1024 THEN 'MEDIUM'
    ELSE 'OK'
  END AS priority
FROM information_schema.tables
WHERE table_schema = 'test_backup_restore';

-- 4. OPTIMIZE (only if fragmentation > 10MB)
SELECT '4. OPTIMIZE TABLE:' AS phase;
SET @frag_mb = (SELECT data_free/1024/1024 FROM information_schema.tables
                WHERE table_schema='test_backup_restore' AND table_name='employees');
SET @optimize_needed = IF(@frag_mb > 10, 'YES', 'NO');
SELECT @optimize_needed AS optimize_employees, @frag_mb AS fragmentation_mb;

OPTIMIZE TABLE IF(@frag_mb > 10, test_backup_restore.employees, NULL);

-- 5. POST-MAINTENANCE VALIDATION
SELECT '5. Post-Maintenance Status:' AS phase;
CHECK TABLE test_backup_restore.employees QUICK;
CHECK TABLE test_backup_restore.departments QUICK;
```

```
-- 6. PERFORMANCE SUMMARY
SELECT '6. Maintenance Complete:' AS phase, NOW() AS end_time,
    TIMEDIFF(NOW(), @maintenance_start) AS duration;

SELECT 'SUCCESS: All maintenance operations completed' AS final_status;
```

**Step 2: Production-grade bash orchestrator**

**File: /opt/mysql_maintenance/master_maintenance.sh**

```
#!/bin/bash
# ============================================================
# MySQL Master Maintenance Orchestrator
# ============================================================

export MYSQL_ROOT_PASS='yourpassword'
LOG_DIR="/var/log/mysql_maintenance"
DATE=$(date +%Y%m%d_%H%M%S)
BACKUP_DIR="/backup/maintenance"

# Create directories
mkdir -p "$LOG_DIR" "$BACKUP_DIR"

LOG_FILE="${LOG_DIR}/master_maintenance_${DATE}.log"
PRE_MAINTENANCE="${LOG_DIR}/pre_${DATE}.txt"
POST_MAINTENANCE="${LOG_DIR}/post_${DATE}.txt"

echo "=== MySQL Master Maintenance Started: $(date) ===" | tee $LOG_FILE
```

```
# 0. PRE-CHECKS

echo "PHASE 0: Pre-maintenance checks..." | tee -a $LOG_FILE

mysql -u root -p$MYSQL_ROOT_PASS -N -e "

SELECT @@hostname, VERSION(), NOW();

SHOW GLOBAL STATUS LIKE 'Threads_connected';

" > $PRE_MAINTENANCE


# 1. QUICK BACKUP SNAPSHOT (safety first)

echo "PHASE 1: Quick backup snapshot..." | tee -a $LOG_FILE

mysqldump -u root -p$MYSQL_ROOT_PASS --single-transaction --routines --triggers \

  test_backup_restore | gzip > ${BACKUP_DIR}/pre_maint_${DATE}.sql.gz


# 2. RUN COMPREHENSIVE MAINTENANCE

echo "PHASE 2: Running comprehensive maintenance..." | tee -a $LOG_FILE

mysql -u root -p$MYSQL_ROOT_PASS <
/opt/mysql_maintenance/comprehensive_maintenance.sql \

  >> $LOG_FILE 2>&1


# 3. POST-VERIFICATION

echo "PHASE 3: Post-maintenance verification..." | tee -a $LOG_FILE

mysql -u root -p$MYSQL_ROOT_PASS -N -e "

SELECT

  table_name, engine,

  ROUND(data_free/1024/1024,2) AS post_frag_mb

FROM information_schema.tables

WHERE table_schema='test_backup_restore';

" > $POST_MAINTENANCE
```

```
# 4. COMPARE AND ALERT
echo "PHASE 4: Results analysis..." | tee -a $LOG_FILE
python3 /opt/mysql_maintenance/analyze_results.py $PRE_MAINTENANCE
$POST_MAINTENANCE $LOG_FILE

# 5. CLEANUP OLD LOGS (keep 30 days)
find $LOG_DIR -name "*.log" -mtime +30 -delete
find $BACKUP_DIR -name "*.sql.gz" -mtime +30 -delete

echo "=== Maintenance COMPLETED: $(date) ===" | tee -a $LOG_FILE
```

**Step 3: Results analysis script**

**File: /opt/mysql_maintenance/analyze_results.py**

```
#!/usr/bin/env python3
import sys
import re

pre_file, post_file, log_file = sys.argv[1], sys.argv[2], sys.argv[3]

# Extract fragmentation from post-maintenance
with open(post_file, 'r') as f:
    post_data = f.read()

frag_match = re.search(r'(\w+)\s+\w+\s+([\d.]+)', post_data)
if frag_match:
    table, frag_mb = frag_match.groups()
    print(f"FINAL STATUS: {table} fragmentation: {frag_mb}MB", file=open(log_file, 'a'))

# Alert if high fragmentation remains
if float(frag_mb or 0) > 10:
    print("ALERT: High fragmentation remains after maintenance!",
          file=open(log_file, 'a'))
```

**Step 4: Multi-tier scheduling**

**Cron configuration (crontab -e):**

```
# Daily light maintenance (2AM - 15 mins)
0 2 * * * /opt/mysql_maintenance/master_maintenance.sh -light

# Weekly comprehensive (Sunday 3AM - 45 mins)
0 3 * * 0 /opt/mysql_maintenance/master_maintenance.sh -full

# Monthly deep maintenance (1st, 1AM - 2 hours)
0 1 1 * * /opt/mysql_maintenance/master_maintenance.sh -deep
```

**Step 5: Execution and verification**

**Test run:**

```
chmod +x /opt/mysql_maintenance/*.sh
/opt/mysql_maintenance/master_maintenance.sh
tail -50 /var/log/mysql_maintenance/master_maintenance_*.log
```

**Expected log output:**

```
=== MySQL Master Maintenance Started: Wed Jan 21 02:00:01 IST 2026 ===

PHASE 0: Pre-maintenance checks...

PHASE 1: Quick backup snapshot...

PHASE 2: Running comprehensive maintenance...

3. Fragmentation Analysis:

employees  InnoDB  15.2K  12.45  18.7%  HIGH

4. OPTIMIZE TABLE: YES  12.45MB

6. Maintenance Complete: 00:12:34 duration

FINAL STATUS: employees fragmentation: 2.1MB
```

**Maintenance Operations Matrix**

| Operation | Purpose | Frequency | Locks Tables? |
|---|---|---|---|
| CHECK TABLE | Detect corruption | Daily | Yes (brief) |
| ANALYZE TABLE | Update index statistics | Daily | Yes (brief) |
| OPTIMIZE | Defragment + rebuild | Weekly | Yes (long) |
| Backup Snap | Safety before changes | Every run | Online (InnoDB) |

**Production Checklist**

- Test during low-traffic window first
- Verify backup restoration works
- Monitor data_free reduction post-maintenance
- Set up log rotation for maintenance logs
- Alert only on actual failures

This creates enterprise-grade maintenance combining all key operations with safety checks, logging, and alerting.