

# Performance Tuning & Monitoring

## 9.1 Query Optimization and Indexing

### Query Execution Plans with EXPLAIN

#### EXPLAIN Output Columns:

EXPLAIN SELECT \* FROM employees WHERE department = 'Sales'\G

*-- Key columns:*

*-- id: SELECT sequence*

*-- select\_type: Type of SELECT (SIMPLE, PRIMARY, SUBQUERY, etc.)*

*-- table: Table accessed*

*-- partitions: Matched partitions*

*-- type: How table is accessed (see below)*

*-- possible\_keys: Indexes that could be used*

*-- key: Actual index used*

*-- key\_len: Index length*

*-- ref: Column comparison*

*-- rows: Estimated rows examined*

*-- filtered: Filtered by WHERE condition*

*-- Extra: Additional information*

#### Type Values (Best to Worst):

- **system:** Only one row (constant)
- **const:** One match (PRIMARY KEY or UNIQUE)
- **eq\_ref:** One row per input (foreign key)
- **ref:** Multiple rows (non-unique index)
- **range:** Index range scan (WHERE with <, >, BETWEEN)
- **index:** Full index scan

- **ALL:** Full table scan (worst)

## Creating Effective Indexes

### Single Column Index:

*-- Indexes columns used in WHERE, JOIN, ORDER BY, GROUP BY*

```
CREATE INDEX idx_email ON users(email);
```

*-- Verify index usage*

```
EXPLAIN SELECT * FROM users WHERE email = 'john@example.com'\G
```

### Composite Index (Multi-column):

*-- Order matters: WHERE conditions first, then ORDER BY*

```
CREATE INDEX idx_status_date ON orders(status, created_at);
```

*-- Queries that can use this index:*

*-- 1. WHERE status = 'completed'*

*-- 2. WHERE status = 'completed' AND created\_at > '2026-01-01'*

*-- 3. WHERE status = 'completed' ORDER BY created\_at*

*-- Query that cannot use index:*

*-- SELECT \* WHERE created\_at > '2026-01-01' -- Wrong order*

### Full-Text Index:

```
CREATE FULLTEXT INDEX idx_content ON articles(content);
```

*-- Full-text search*

```
SELECT * FROM articles WHERE MATCH(content) AGAINST('database' IN BOOLEAN MODE);
```

### **Prefix Index (for large columns):**

*-- Index only first N characters*

CREATE INDEX idx\_url ON websites(url(100)); *-- Index first 100 chars*

### **Identifying Slow Queries**

*-- Enable slow query log*

SET GLOBAL slow\_query\_log = 'ON';

SET GLOBAL long\_query\_time = 1; *-- Log queries > 1 second*

*-- Force slow query (for testing)*

SELECT SLEEP(2);

*-- View slow query log*

mysql -u root -p -e "SELECT \* FROM mysql.slow\_log ORDER BY start\_time DESC LIMIT 10\G"

*-- Parse slow query log*

mysqldumpslow /var/log/mysql/slow-query.log | head -20

### **Query Optimization Tips**

**\*\*Tip 1: Avoid SELECT \*\***

*-- Bad: Unnecessary columns*

SELECT \* FROM orders WHERE order\_id = 1;

*-- Good: Only needed columns*

SELECT order\_id, customer\_id, total FROM orders WHERE order\_id = 1;

### Tip 2: Use LIMIT with Offset Carefully

*-- Bad: Full scan with offset*

```
SELECT * FROM large_table ORDER BY id DESC LIMIT 1000, 20;
```

*-- Good: Use WHERE with indexed column*

```
SELECT * FROM large_table WHERE id < 1000 ORDER BY id DESC LIMIT 20;
```

### Tip 3: Avoid Functions in WHERE Clause

*-- Bad: Function in WHERE prevents index use*

```
SELECT * FROM orders WHERE YEAR(created_at) = 2026;
```

*-- Good: Range condition*

```
SELECT * FROM orders WHERE created_at >= '2026-01-01' AND created_at < '2027-01-01';
```

### Tip 4: Use UNION ALL Instead of UNION

*-- Bad: UNION removes duplicates (slower)*

```
SELECT id FROM customers UNION SELECT id FROM suppliers;
```

*-- Good: If duplicates acceptable*

```
SELECT id FROM customers UNION ALL SELECT id FROM suppliers;
```

### Tip 5: Join Optimization

*-- Consider join order (smallest table first)*

```
SELECT c.customer_id, c.name, COUNT(o.order_id) as order_count
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id
WHERE c.country = 'USA'
GROUP BY c.customer_id;
```

*-- Ensure foreign key columns are indexed*

```
CREATE INDEX idx_customer_id ON orders(customer_id);
```

## **9.2 Performance Schema Monitoring**

### **Enabling Performance Schema**

*-- Check if enabled*

```
SHOW VARIABLES LIKE 'performance_schema'; -- Should be ON
```

*-- View current instrumentation*

```
SELECT * FROM performance_schema.setup_instruments;
```

*-- Enable specific instruments*

```
UPDATE performance_schema.setup_instruments
```

```
SET ENABLED = 'YES', TIMED = 'YES'
```

```
WHERE NAME LIKE 'statement/%' OR NAME LIKE 'wait/%';
```

*-- Enable consumers*

```
UPDATE performance_schema.setup_consumers
```

```
SET ENABLED = 'YES'
```

```
WHERE NAME LIKE '%statements%' OR NAME LIKE '%events%';
```

### **Table I/O Statistics**

*-- Find most accessed tables*

```
SELECT
```

```
    OBJECT_SCHEMA,
```

```
    OBJECT_NAME,
```

```
    COUNT_READ,
```

```
COUNT_WRITE,
COUNT_INSERT,
COUNT_UPDATE,
COUNT_DELETE,
COUNT_READ + COUNT_WRITE AS total_access
FROM performance_schema.table_io_waits_summary_by_table
WHERE OBJECT_SCHEMA NOT IN ('mysql', 'performance_schema', 'information_schema')
ORDER BY total_access DESC
LIMIT 10\G
```

### **Index Usage Statistics**

*-- Find unused indexes*

```
SELECT
    OBJECT_SCHEMA,
    OBJECT_NAME,
    INDEX_NAME,
    COUNT_READ,
    COUNT_WRITE
FROM performance_schema.table_io_waits_summary_by_index_usage
WHERE COUNT_READ = 0 AND COUNT_WRITE = 0
AND OBJECT_SCHEMA NOT IN ('mysql', 'performance_schema')\G
```

*-- Find frequently used indexes*

```
SELECT
    OBJECT_SCHEMA,
    OBJECT_NAME,
    INDEX_NAME,
    COUNT_READ
```

```
FROM performance_schema.table_io_waits_summary_by_index_usage
WHERE COUNT_READ > 0
ORDER BY COUNT_READ DESC
LIMIT 10\G
```

### **Query Statistics**

*-- Top queries by execution count*

```
SELECT
    EVENT_NAME,
    COUNT_STAR,
    SUM_TIMER_WAIT / 1000000000 AS total_wait_ms,
    AVG_TIMER_WAIT / 1000000000 AS avg_wait_ms
FROM performance_schema.events_statements_summary_by_event_name
ORDER BY COUNT_STAR DESC
LIMIT 10\G
```

*-- Top queries by total wait time*

```
SELECT
    DIGEST_TEXT,
    COUNT_STAR,
    SUM_TIMER_WAIT / 1000000000000 AS total_wait_sec,
    AVG_TIMER_WAIT / 1000000000 AS avg_wait_ms
FROM performance_schema.events_statements_summary_by_digest
ORDER BY SUM_TIMER_WAIT DESC
LIMIT 10\G
```

## Lock Contention

*-- Find tables with lock waits*

```
SELECT
    OBJECT_SCHEMA,
    OBJECT_NAME,
    COUNT_STAR,
    SUM_TIMER_WAIT / 1000000000000 AS total_wait_sec
FROM performance_schema.table_lock_waits_summary_by_table
WHERE COUNT_STAR > 0\G
```

*-- Current lock waits*

```
SELECT
    REQUESTING_THREAD_ID,
    BLOCKING_THREAD_ID,
    OBJECT_SCHEMA,
    OBJECT_NAME
FROM performance_schema.table_lock_waits_summary_by_table\G
```

## 9.3 Buffer Pool Configuration and Monitoring

### Sizing InnoDB Buffer Pool

#### Rule of Thumb:

- 60-80% of available RAM
- Example: 16GB server → 10-12GB buffer pool

[mysqld]

innodb\_buffer\_pool\_size = 10G

*# Multiple pool instances for better concurrency*

innodb\_buffer\_pool\_instances = 8

*# Chunk size for memory allocation*

innodb\_buffer\_pool\_chunk\_size = 128M

### **Monitoring Buffer Pool**

*-- Buffer pool statistics*

SELECT

POOL\_ID,

POOL\_SIZE,

FREE\_BUFFERS,

DATABASE\_PAGES,

DIRTY\_PAGES,

PAGES\_MADE\_YOUNG,

PAGES\_NOT\_MADE\_YOUNG,

ROUND(((DATABASE\_PAGES / POOL\_SIZE) \* 100), 2) AS usage\_pct

FROM INFORMATION\_SCHEMA.INNODB\_BUFFER\_POOL\_STATS;

*-- Calculate hit ratio*

*-- (buffer\_pool\_reads - buffer\_pool\_read\_requests) / buffer\_pool\_read\_requests \* 100*

SHOW STATUS LIKE 'Innodb\_buffer\_pool%';

**Ideal Hit Ratio: > 99%**

## 9.4 Table Partitioning Strategies

### Range Partitioning

Useful for time-series data:

```
CREATE TABLE sales (  
    sale_id INT PRIMARY KEY AUTO_INCREMENT,  
    sale_date DATE,  
    amount DECIMAL(10, 2)  
) ENGINE=InnoDB  
PARTITION BY RANGE (YEAR(sale_date)) (  
    PARTITION p2020 VALUES LESS THAN (2021),  
    PARTITION p2021 VALUES LESS THAN (2022),  
    PARTITION p2022 VALUES LESS THAN (2023),  
    PARTITION p2023 VALUES LESS THAN (2024),  
    PARTITION p_future VALUES LESS THAN MAXVALUE  
);
```

*-- Add new partition*

```
ALTER TABLE sales ADD PARTITION (PARTITION p2024 VALUES LESS THAN (2025));
```

*-- Drop old partition*

```
ALTER TABLE sales DROP PARTITION p2020;
```

## Hash Partitioning

Distribute data evenly:

```
CREATE TABLE customer_data (  
    customer_id INT PRIMARY KEY,  
    name VARCHAR(255),  
    balance DECIMAL(10, 2)  
) ENGINE=InnoDB  
  
PARTITION BY HASH (customer_id) PARTITIONS 4;
```

## Benefits of Partitioning

1. **Faster Queries:** Partition pruning skips irrelevant partitions
2. **Easier Maintenance:** Drop/archive old partitions
3. **Parallel Execution:** Multi-threaded queries across partitions
4. **Better Memory:** Smaller partitions fit better in cache

## 9.5 Monitoring Tools Overview

### Workbench

GUI tool for:

- Connection management
- Query execution and analysis
- Schema design
- Performance monitoring

### Prometheus + Grafana

For containerized/cloud environments:

*# Install MySQL exporter*

```
docker run prom/mysqld-exporter \  
    --config.my-cnf=/etc/mysql/my.cnf
```

*# Create Grafana dashboards*

*# (Predefined dashboards available)*

## **ELK Stack Integration**

Send MySQL logs to Elasticsearch:

*# Filebeat config*

filebeat.inputs:

- type: log

enabled: true

paths:

- /var/log/mysql/slow-query.log

multiline.pattern: '^# Time:'

multiline.negate: true

multiline.match: after

output.elasticsearch:

hosts: ["localhost:9200"]

## **9.6 Summary: Key Takeaways**

1. **Query Optimization:** Use EXPLAIN, create indexes, avoid functions in WHERE
2. **Indexing Strategy:** Composite indexes with correct column order
3. **Performance Schema:** Monitor table I/O, index usage, query statistics
4. **Buffer Pool:** Size 60-80% of RAM, monitor hit ratio (>99%)
5. **Partitioning:** Range/hash partitioning for large tables
6. **Monitoring Tools:** Workbench, Prometheus/Grafana, ELK Stack
7. **Best Practices:** Regular ANALYZE, monitor slow queries, test performance changes