MySQL

Interview Questions and Answers

Core Concepts

This section focuses on fundamental principles and advanced concepts that an experienced developer should master.

1. Explain MySQL's MVCC (Multi-Version Concurrency Control) implementation and its benefits

MVCC in MySQL's InnoDB implements concurrency control by maintaining multiple versions of rows:

- Each transaction sees a snapshot of data as it existed when the transaction started
- Uses undo logs to maintain previous versions of records
- Implements consistent reads without locking

Key Benefits:

- Better read scalability as readers don't block writers
- Consistent transaction isolation
- Eliminates most read-write conflicts

2. How would you optimize a slow-performing MySQL query?

Systematic Approach to Query Optimization:

- Use EXPLAIN to analyze query execution plan
- Check for proper indexing strategy
- Verify if all JOINs are necessary
- Consider query rewriting

Example optimization:

```
-- Before
SELECT * FROM orders o
JOIN users u ON u.id = o.user_id
WHERE o.status = 'pending';

-- After
SELECT o.id, o.amount, u.name
FROM orders o
FORCE INDEX (idx_status)
JOIN users u ON u.id = o.user_id
WHERE o.status = 'pending';
```

3. Describe MySQL's replication mechanisms and their use cases

MySQL Replication Types:

- **Asynchronous Replication:** Standard primary-secondary setup
- **Semi-synchronous:** Primary waits for at least one secondary acknowledgment
- **Group Replication:** Multi-primary or single-primary with automatic failover

Use Cases:

- Scalable read operations
- Disaster recovery
- Geographic distribution
- Real-time analytics on replicas

4. How does MySQL handle deadlock detection and resolution?

Deadlock Handling in MySQL:

- InnoDB automatically detects deadlocks using wait-for graph algorithm
- Rolls back transaction with smaller weight (usually newer)
- Maintains deadlock detection thread

Prevention Strategies:

- Access tables in consistent order
- Keep transactions short
- Use appropriate isolation level

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SELECT ... FOR UPDATE NOWAIT;
```

5. Explain MySQL's query cache mechanism and when to avoid it

Query Cache (Deprecated in MySQL 8.0):

- Stores complete result sets with exact SQL text
- Invalidated when underlying tables change

Reasons to Avoid:

- High overhead for cache maintenance
- Frequent invalidation in write-heavy workloads
- Global mutex contention

Better Alternatives:

- Application-level caching
- ProxySQL query caching
- Proper indexing strategy

6. How would you implement database partitioning in MySQL?

Partitioning Methods:

- **RANGE:** Based on value ranges
- **LIST:** Based on discrete values
- **HASH:** Using hashing function
- **KEY:** Similar to HASH but MySQL manages

```
CREATE TABLE sales (
  id INT,
  sale_date DATE
) PARTITION BY RANGE (YEAR(sale_date)) (
  PARTITION p2021 VALUES LESS THAN (2022),
  PARTITION p2022 VALUES LESS THAN (2023)
);
```

7. Describe MySQL's InnoDB buffer pool and its optimization techniques

Buffer Pool Components:

- Data pages cache
- Index pages cache
- Change buffer
- Adaptive hash index

Optimization Techniques:

- Proper sizing (typically 80% of available RAM)
- Multiple buffer pool instances
- Buffer pool preloading

```
SET GLOBAL innodb_buffer_pool_size = 12884901888;
SET GLOBAL innodb_buffer_pool_instances = 8;
```

8. How do you implement and maintain database security in MySQL?

Security Layers:

- **Authentication:** Strong password policies, SSL/TLS
- **Authorization:** GRANT/REVOKE privileges
- **Encryption:** Data at rest and in transit

```
CREATE USER 'app_user'@'%'
IDENTIFIED BY 'complex_password'
REQUIRE SSL;
GRANT SELECT, INSERT ON app_db.*
TO 'app_user'@'%';
```

9. Explain MySQL's gap locks and their importance in transaction isolation

Gap Locks in InnoDB:

- Prevent phantom reads in REPEATABLE READ
- Lock ranges between existing index records
- Part of next-key locking strategy

Important Considerations:

- Only used for certain isolation levels
- Can cause concurrency issues
- Critical for maintaining consistency

```
SELECT * FROM orders
WHERE id BETWEEN 100 AND 200
FOR UPDATE;
```

10. How would you design a high-availability MySQL architecture?

Key Components:

- **Primary-Secondary Replication:** Automated failover
- **Load Balancing:** ProxySQL or MySQL Router
- **Monitoring:** Performance metrics and alerts

Best Practices:

- Geographic distribution
- Regular backup verification
- Automated health checks
- Orchestration tools (Orchestrator)

Data Structures and Algorithms

Questions in this section test your understanding of how to work with and manipulate data efficiently.

1. How would you implement a sliding window analysis for time-series data in MySQL?

Sliding Window Implementation

Key Concepts:

- Window function usage
- Range-based partitioning

```
SELECT timestamp, value,
AVG(value) OVER (
  ORDER BY timestamp
  RANGE BETWEEN INTERVAL '5' MINUTE PRECEDING
  AND CURRENT ROW
) AS moving_avg
```

2. How would you implement an LRU (Least Recently Used) Cache in MySQL?

LRU Cache Implementation

Key Components:

- Use a doubly-linked list for O(1) removal/insertion
- HashMap for O(1) lookups
- Fixed capacity enforcement

```
CREATE TABLE lru_cache (
  key_id VARCHAR(255) PRIMARY KEY,
  value TEXT,
  access_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE INDEX idx_access_time ON lru_cache(access_time);
```

3. Explain how you would efficiently find pairs of numbers that sum to a target value in a large MySQL table?

Two-Sum Problem Solution

Efficient Approach:

- Use indexing for better performance
- Single table scan with hash join

```
SELECT a.number AS num1, b.number AS num2
FROM numbers a JOIN numbers b
WHERE a.id < b.id
AND a.number + b.number = target_sum;
```

4. Explain how to implement a hierarchical data structure (tree) in MySQL?

Tree Implementation Approaches

Common Methods:

- Adjacency List Model
- Nested Set Model

- Closure Table

```
CREATE TABLE tree_nodes (
  id INT PRIMARY KEY,
  parent_id INT,
  name VARCHAR(255),
  FOREIGN KEY (parent_id) REFERENCES tree_nodes(id)
);
```

5. How would you implement a queue data structure using MySQL tables?

Queue Implementation

Essential Features:

- FIFO operations
- Atomic operations
- Race condition prevention

```
CREATE TABLE queue (
  id INT AUTO_INCREMENT PRIMARY KEY,
  payload JSON,
  status ENUM('pending','processing','complete'),
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

6. Explain how to implement a graph database structure in MySQL?

Graph Structure Implementation

Components:

- Vertices table
- Edges table
- Properties handling

```
CREATE TABLE edges (
  from_node INT,
  to_node INT,
  weight FLOAT,
  PRIMARY KEY (from_node, to_node)
);
```

7. How would you implement a priority queue in MySQL?

Priority Queue Implementation

Key Features:

- Priority-based ordering
- Efficient insertion/deletion
- Index optimization

```
CREATE TABLE priority_queue (
  id INT AUTO_INCREMENT,
  priority INT,
  data JSON,
  PRIMARY KEY (priority, id)
);
```

8. Explain how to implement a hash table with collision handling in MySQL?

Hash Table Implementation

Collision Handling:

- Separate chaining
- Linear probing
- Bucket management

```
CREATE TABLE hash_table (
  bucket_id INT,
  key_hash INT,
  value JSON,
  next_pointer INT,
  PRIMARY KEY (bucket_id, key_hash)
);
```

9. How would you implement a trie (prefix tree) structure in MySQL?

Trie Implementation

Structure Components:

- Node representation
- Character transitions
- Word marking

```
CREATE TABLE trie_nodes (
  node_id INT PRIMARY KEY,
  character CHAR(1),
  parent_id INT,
  is_word BOOLEAN
);
```

10. Explain how to implement a stack with push/pop operations in MySQL?

Stack Implementation

Operations:

- LIFO functionality
- Atomic operations
- Transaction safety

```
CREATE TABLE stack (
  id INT AUTO_INCREMENT PRIMARY KEY,
  value JSON,
  pushed_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  INDEX (pushed_at)
);
```

System Design

These questions evaluate your ability to think about the bigger picture, including architecture, scalability, and performance.

1. How would you design a scalable URL shortener service like bit.ly?

Key Components:

- **Data Storage**: MySQL for storing URL mappings with indexed short codes
- **Cache Layer**: Redis for frequently accessed URLs
- **Load Balancer**: For distributing traffic across application servers

Database Schema:

```
CREATE TABLE urls (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  short_code VARCHAR(7) UNIQUE,
  long_url VARCHAR(2048),
  created_at TIMESTAMP,
  clicks INT DEFAULT 0
);
```

Key Considerations:

- Base62 encoding for generating short codes
- Cache eviction policy (LRU)
- Rate limiting to prevent abuse
- Analytics tracking

2. Design a social media feed system that can handle millions of users

Architecture Components:

- **Storage**: Sharded MySQL for user and post data
- **Cache**: Redis for feed composition
- **Message Queue**: Kafka for async processing
- **CDN**: For media content delivery

Schema Design:

```
CREATE TABLE posts (
  post_id BIGINT PRIMARY KEY,
  user_id BIGINT,
  content TEXT,
  created_at TIMESTAMP,
  INDEX(user_id, created_at)
);
```

Key Features:

- Feed pagination
- Pull vs Push model for feed updates
- Content ranking algorithms
- Real-time notifications

3. How would you implement a distributed rate limiter using MySQL?

Implementation Approach:

- **Token Bucket Algorithm** with distributed counters
- **Sliding Window** implementation

```
CREATE TABLE rate_limits (
  key_id VARCHAR(255),
  window_start TIMESTAMP,
  request_count INT,
  PRIMARY KEY (key_id, window_start)
);
```

Considerations:

- Race conditions handling with transactions
- Cleanup of expired windows
- High availability requirements
- Monitoring and alerting

4. Design a real-time chat system that scales to millions of concurrent users

Architecture Components:

- **WebSocket Servers** for real-time communication
- **MySQL** for message persistence
- **Redis** for presence management

```
CREATE TABLE messages (
  msg_id BIGINT PRIMARY KEY,
  chat_id BIGINT,
  sender_id BIGINT,
  content TEXT,
  sent_at TIMESTAMP
);
```

Key Features:

- Message delivery guarantees
- Online/offline status
- Message history
- Group chat support

5. How would you design a notification system for a large-scale application?

System Components:

- **Event Producer**: Application services
- **Message Queue**: Kafka for event distribution
- **Storage**: MySQL for notification persistence

```
CREATE TABLE notifications (
  id BIGINT PRIMARY KEY,
  user_id BIGINT,
  type ENUM('email','push','sms'),
  status VARCHAR(20),
  created_at TIMESTAMP
);
```

Considerations:

- Delivery retry mechanism
- Template management
- Priority queuing
- Rate limiting

6. Design a distributed task scheduler system

Core Components:

- **Job Store**: MySQL for task definitions

- **Distributed Lock**: For leader election
- **Worker Nodes**: For task execution

```
CREATE TABLE scheduled_tasks (
  task_id BIGINT PRIMARY KEY,
  cron_expression VARCHAR(100),
  task_data JSON,
  next_run TIMESTAMP
);
```

Features:

- Fault tolerance
- Task prioritization
- Dead job detection
- Monitoring dashboard

7. How would you design a leaderboard system for a gaming application?

Architecture:

- **Storage**: MySQL for historical data
- **Cache**: Redis Sorted Sets for real-time rankings
- **Batch Processing**: For periodic updates

```
CREATE TABLE scores (
  user_id BIGINT,
  score BIGINT,
  timestamp TIMESTAMP,
  INDEX(score DESC)
);
```

Key Features:

- Real-time updates
- Time-based rankings
- Global and regional rankings
- Friends-only leaderboards

8. Design a distributed session management system

Components:

- **Session Store**: MySQL + Redis
- **Load Balancer**: Sticky sessions
- **Authentication Service**

```
CREATE TABLE sessions (
  session_id VARCHAR(64) PRIMARY KEY,
  user_id BIGINT,
  data JSON,
  expires_at TIMESTAMP
);
```

Considerations:

- Session replication
- Cleanup strategy
- Security measures
- High availability

9. How would you design a payment processing system?

Core Components:

- **Transaction Store**: MySQL with ACID compliance
- **Queue**: For async processing
- **State Machine**: For transaction flow

```
CREATE TABLE transactions (
  tx_id BIGINT PRIMARY KEY,
  status ENUM('pending','completed','failed'),
  amount DECIMAL(10,2),
  created_at TIMESTAMP
);
```

Key Features:

- Idempotency
- Transaction logging
- Retry mechanism
- Fraud detection

10. Design a content delivery system for streaming video

System Components:

- **Metadata Store**: MySQL for video information
- **CDN**: For content delivery
- **Transcoding Service**: For format conversion

```
CREATE TABLE videos (
  video_id BIGINT PRIMARY KEY,
  status VARCHAR(20),
  duration INT,
  formats JSON
);
```

Features:

- Adaptive bitrate streaming
- Content protection
- Analytics tracking
- Cache optimization

Coding and Debugging

This section presents practical coding challenges and questions about debugging techniques.

1. How would you optimize a slow performing MySQL query? Show an example of query optimization.

Key Optimization Techniques:

- Use EXPLAIN to analyze query execution plan
- Add proper indexes
- Avoid SELECT *
- Use WHERE conditions efficiently

Example:

```
-- Slow Query
SELECT * FROM orders JOIN users
WHERE orders.status = 'pending';

-- Optimized Query
SELECT o.order_id, o.amount, u.name
FROM orders o
INNER JOIN users u ON o.user_id = u.id
WHERE o.status = 'pending'
INDEX(status);
```

2. Write a MySQL query to find duplicate records in a table.

```
SELECT name, email, COUNT(*) as count
FROM users
GROUP BY name, email
HAVING COUNT(*) > 1;
```

Key Points:

- GROUP BY groups identical records
- HAVING filters groups with count > 1
- Can be modified to include more columns

3. How would you implement a soft delete mechanism in MySQL?

Implementation Approach:

```
ALTER TABLE users
ADD deleted_at TIMESTAMP NULL DEFAULT NULL;

-- Instead of DELETE:
UPDATE users
SET deleted_at = CURRENT_TIMESTAMP
WHERE id = ?;
```

- Add deleted_at timestamp column
- Modify queries to check: WHERE deleted_at IS NULL
- Maintains referential integrity
- Allows data recovery

4. Write a query to find the nth highest salary from an employees table.

```
SELECT DISTINCT salary
FROM employees e1
```

```
WHERE N-1 = (
  SELECT COUNT(DISTINCT salary)
  FROM employees e2
  WHERE e2.salary > e1.salary
);
```

Alternative using LIMIT:

```
SELECT salary
FROM employees
ORDER BY salary DESC
LIMIT N-1, 1;
```

5. How would you handle deadlocks in MySQL? Show an example transaction.

Deadlock Prevention Strategy:

- Order transactions consistently
- Keep transactions short
- Use appropriate isolation level

```
START TRANSACTION;
SELECT * FROM accounts WHERE id = 1 FOR UPDATE;
UPDATE accounts SET balance = balance - 100 WHERE id = 1;
COMMIT;
```

Best Practices:

- Use innodb_deadlock_detect
- Set innodb_lock_wait_timeout

6. Write a query to pivot rows into columns dynamically.

```
SET @sql = NULL;
SELECT GROUP_CONCAT(
  DISTINCT CONCAT(
    'MAX(IF(month = ''', month, ''', amount, NULL)) AS ', month
  )
) INTO @sql FROM sales;
SET @sql = CONCAT('SELECT year, ', @sql, ' FROM sales GROUP BY year');
PREPARE stmt FROM @sql;
EXECUTE stmt;
```

7. How would you implement hierarchical data in MySQL? Show an example.

Nested Set Model:

```
CREATE TABLE categories (
  id INT PRIMARY KEY,
  name VARCHAR(100),
  lft INT,
  rgt INT,
  INDEX(lft, rgt)
);
```

Query for all children:

```
SELECT child.* FROM categories AS node
JOIN categories AS child ON child.lft BETWEEN node.lft AND node.rgt
WHERE node.id = 1;
```

8. Write a query to calculate running totals efficiently.

Using Window Functions:

```
SELECT
  date,
  amount,
  SUM(amount) OVER (
```

```
    ORDER BY date
    ROWS UNBOUNDED PRECEDING
  ) as running_total
FROM transactions;
```

Note: Window functions are more efficient than correlated subqueries for running totals.

9. How would you implement full-text search in MySQL?

Setup:

```
ALTER TABLE articles
ADD FULLTEXT(title, content);

-- Search Query
SELECT * FROM articles
WHERE MATCH(title, content)
AGAINST('search terms' IN BOOLEAN MODE);
```

- Use InnoDB or MyISAM
- Consider stopwords
- Use IN NATURAL LANGUAGE MODE for relevance

10. Write a query to find gaps in sequential data.

```
SELECT seq.id + 1 as gap_start,
       next_seq.id - 1 as gap_end
FROM sequence seq
LEFT JOIN sequence next_seq
  ON seq.id + 1 = next_seq.id
WHERE next_seq.id IS NULL;
```

Use Cases:

- Finding missing invoice numbers
- Identifying sequence breaks
- Audit trail validation

Behavioral Questions

These questions assess your soft skills, problem-solving approach, and how you work in a team.

1. Tell me about a time when you optimized a poorly performing MySQL query.

Situation: At my previous company, we had a dashboard query taking 15+ seconds to execute, causing timeout issues.

Task: I needed to optimize the query to execute in under 2 seconds to meet our SLA requirements.

Action: I:

- Used EXPLAIN to analyze query execution plan
- Added composite indexes on frequently filtered columns
- Rewrote subqueries as JOINs
- Implemented query result caching

Result: Query execution time reduced to 800ms, eliminating timeouts and improving dashboard load times by 90%.

2. Describe a situation where you had to implement a complex database migration without downtime.

Situation: We needed to split a monolithic users table into normalized customer and address tables.

Task: Migrate 10M+ records with zero downtime during business hours.

Action: I:

- Created new tables and dual-write functionality
- Wrote incremental migration scripts
- Implemented feature flags for gradual rollout
- Monitored replication lag throughout

Result: Successfully migrated all data over 2 weeks with no service interruptions or data loss.

3. Tell me about a time you had to debug a critical production database issue.

Situation: Production database suddenly experienced 100% CPU usage during peak hours.

Task: Identify and resolve the root cause while maintaining system availability.

Action: I:

- Analyzed slow query logs and process list
- Identified rogue queries from new feature
- Implemented query throttling
- Added missing indexes

Result: CPU usage normalized within 30 minutes, implemented monitoring alerts to prevent future incidents.

4. Share an experience where you had to make a difficult technical decision regarding database architecture.

Situation: Team debated between master-slave vs multi-master replication for global expansion.

Task: Evaluate options and recommend the best architecture for our use case.

Action: I:

- Created decision matrix with pros/cons
- Built proof-of-concept implementations
- Conducted performance testing
- Presented findings to stakeholders

Result: Successfully implemented master-slave with regional read replicas, reducing latency by 65%.

5. Describe a time when you had to lead a database upgrade project.

Situation: Required upgrade from MySQL 5.7 to 8.0 for 50+ production databases.

Task: Plan and execute upgrade with minimal disruption.

Action: I:

- Created detailed upgrade runbook
- Tested upgrade in staging environment
- Identified deprecated features and SQL modes
- Coordinated with application teams

Result: Completed upgrade ahead of schedule with only 10 minutes of planned downtime.

6. Tell me about a time you had to improve database security measures.

Situation: Security audit revealed potential vulnerabilities in database access patterns.

Task: Implement enhanced security measures while maintaining functionality.

Action: I:

- Implemented role-based access control
- Enabled SSL for client connections
- Set up audit logging
- Created automated compliance reports

Result: Passed follow-up security audit with zero findings, established new security baseline.

7. Share an experience where you had to handle database capacity planning.

Situation: Rapid user growth threatened to exceed database capacity within 3 months.

Task: Develop and implement scaling strategy.

Action: I:

- Analyzed growth patterns and metrics
- Implemented table partitioning
- Set up automated archiving
- Created capacity forecasting models

Result: Successfully handled 3x growth without performance degradation, established proactive scaling processes.

8. Describe a situation where you had to mentor junior database developers.

Situation: Team hired three junior developers with limited MySQL experience.

Task: Get them productive with our database standards and practices.

Action: I:

- Created learning roadmap
- Conducted weekly training sessions
- Established code review guidelines
- Created documentation templates

Result: All three developers became independent contributors within 3 months, reducing review cycles by 50%.

9. Tell me about a time you had to handle a database disaster recovery situation.

Situation: Production database corruption occurred during peak business hours.

Task: Restore service while minimizing data loss.

Action: I:

- Initiated DR protocol
- Coordinated failover to standby
- Recovered corrupt data from backups
- Communicated status to stakeholders

Result: Restored service within 45 minutes with zero data loss, implemented additional backup verification processes.

10. Share an experience where you had to balance competing database requirements.

Situation: Different teams needed conflicting optimizations for OLTP vs OLAP workloads.

Task: Design solution to satisfy both use cases.

Action: I:

- Analyzed workload patterns
- Implemented read replicas with different configurations
- Set up query routing logic
- Created monitoring dashboards

Result: Achieved 99.9% SLA for OLTP while supporting complex analytics queries, improving both teams' productivity.