

Chapter 33: MySQL Interview & Problem Solving

33.1 Basic MySQL Questions

- * 1. What is MySQL?
- * 2. What is a database? What is a table?
- * 3. What is the difference between **CHAR** and **VARCHAR**?
- * 4. What are primary key and foreign key?
- * 5. What is the difference between **DELETE**, **DROP** and **TRUNCATE**?
- * 6. What is the use of **WHERE** clause?
- * 7. What is the difference between **UNION** and **UNION ALL**?
- * 8. What is normalization? What are its types?
- * 9. What is an index? Why is it used?
- * 10. What is the difference between **INNER JOIN** and **LEFT JOIN**?



Key Points :

- ✓ Understand basic concepts.
- ✓ Know SQL commands.
- ✓ Practice simple queries.
- ✓ Focus on definitions.



These questions test your basic understanding of MySQL concepts.



33.2 Intermediate MySQL Questions

- * 1. What is the difference between clustered and non-clustered index?
- * 2. What are subqueries? Give an example.
- * 3. What is the use of **HAVING** clause?
- * 4. What is the difference between **EXISTS** and **IN**?
- * 5. What is the difference between **SELF JOIN** and **NORMAL JOIN**?
- * 6. What is the use of **GROUP BY** clause?
- * 7. What is the difference between **RANK()**, **DENSE_RANK()** and **ROW_NUMBER()**?
- * 8. What is the purpose of a view?
- * 9. What is a stored procedure? How is it used?
- * 10. What is the difference between **CHAR_LENGTH()** and **LENGTH()**?

Focus Areas :

Subqueries	Working with subqueries
Joins & Groups	Joins, GROUP BY , HAVING
Functions	Built-in functions and their usage
Indexes	Index types and performance

33.3 Advanced MySQL Questions

- * 1. What is an execution plan?
- * 2. How does MySQL optimize queries?
- * 3. What are transactions in MySQL?
- * 4. What is ACID property in transactions?
- * 5. What is the difference between **LOCK** and **UNLOCK**?
- * 6. How does MySQL handle concurrency?
- * 7. What is partitioning in MySQL?
- * 8. What are triggers? How are they used?
- * 9. What is the difference between temporary table and permanent table?
- * 10. How do you improve query performance?

Key Concepts :

- ✓ Query optimization
- ✓ Transactions and ACID
- ✓ Locks and concurrency
- ✓ Partitioning and triggers
- ✓ Performance tuning



These are advanced questions that test your in-depth knowledge of MySQL.



33.4 Query-Based Problems

Problem Type	Description	Example Query
1. Simple Retrieval	Fetch data from a table based on conditions.	<code>SELECT name, age FROM students WHERE age > 20;</code>
2. Aggregation	Use aggregate functions like COUNT() , SUM() , AVG() , MAX() , MIN() .	<code>SELECT COUNT(*) FROM orders WHERE status = 'Shipped';</code>
3. Grouping	Group data using GROUP BY and filter using HAVING .	<code>SELECT department, AVG(salary) FROM employees GROUP BY department HAVING AVG(salary) > 50000;</code>
4. Joins	Combine rows from two or more tables.	<code>SELECT e.name, d.dept_name FROM employees e INNER JOIN department d ON e.dept_id = d.id;</code>
5. Subqueries	Use subqueries in SELECT , WHERE , FROM or HAVING .	<code>SELECT name FROM employees WHERE salary > (SELECT AVG(salary) FROM employees);</code>
6. Advanced	Use window functions, CTEs and complex queries.	<code>SELECT name, salary, RANK() OVER (ORDER BY salary DESC) FROM employees;</code>



Key Takeaway :



Regular practice of interview questions and query problems will build strong MySQL skills. Focus on **understanding concepts**, writing **efficient queries** and **optimizing performance**.

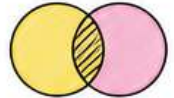


33.5 JOIN Problems

Problem	Description
1. INNER JOIN	Retrieve matching records from two tables.
2. LEFT JOIN	Retrieve all records from left table.
3. RIGHT JOIN	Retrieve all records from right table.
4. FULL JOIN	Retrieve all records when there is a match in either table.
5. SELF JOIN	Join a table with itself.
6. Multi-table JOIN	Join three or more tables and fetch data.

Example Query :

1. `SELECT e.name, d.dept_name
FROM employees e
INNER JOIN departments d ON e.dept_id = d.id;`
2. `SELECT e.name, d.dept_name
FROM employees e
LEFT JOIN departments d ON e.dept_id = d.id;`
3. `SELECT e.name, d.dept_name
FROM employees e
RIGHT JOIN departments d ON e.dept_id = d.id;`
4. `SELECT e.name, d.dept_name
FROM employees e
FULL JOIN departments d ON e.dept_id = d.id;`
5. `SELECT e1.name AS emp, e2.name AS manager
FROM employees e1
JOIN employees e2 ON e1.manager_id = e2.id;`
6. `SELECT o.id, c.name, p.product_name
FROM orders o
JOIN customers c ON o.customer_id = c.id
JOIN products p ON o.product_id = p.id;`



Tables Join



33.6 Subquery Problems

Problem	Description
1. Single Row Subquery	Use subquery that returns a single value.
2. Multiple Row Subquery	Use subquery that returns multiple rows.
3. Multiple Column Subquery	Use subquery that returns multiple columns.
4. Correlated Subquery	Use subquery that depends on the outer query.
5. Subquery with IN / NOT IN	Filter using IN or NOT IN with subquery.
6. Subquery with EXISTS	Check existence of records.

Example Query :

1. `SELECT name FROM employees
WHERE salary = (SELECT MAX(salary) FROM employees);`
2. `SELECT name FROM employees
WHERE dept_id IN (SELECT id FROM departments);`
3. `SELECT name, salary FROM employees
WHERE (dept_id, salary) IN
(SELECT dept_id, MAX(salary) FROM employees
GROUP BY dept_id);`
4. `SELECT name FROM employees e1
WHERE salary > (SELECT AVG(salary)
FROM employees e2
WHERE e1.dept_id = e2.dept_id);`
5. `SELECT name FROM employees
WHERE dept_id NOT IN (SELECT id FROM departments);`
6. `SELECT name FROM employees e
WHERE EXISTS (SELECT 1 FROM orders o
WHERE o.emp_id = e.id);`



Subqueries help in writing powerful and flexible queries.



33.7 CTE Problems

Problem	Description
1. Simple CTE	Use CTE to simplify a query.
2. Multiple CTEs	Use more than one CTE in a query.
3. Recursive CTE	Use recursive CTE for hierarchical data.
4. CTE with Aggregation	Use CTE to perform aggregation.
5. CTE with JOIN	Use CTE and join with other tables.
6. CTE with Filtering	Use CTE and apply filters in main query.

Example Query :

1. `WITH emp_cte AS (SELECT * FROM employees)
SELECT * FROM emp_cte WHERE salary > 50000;`
2. `WITH dept_cte AS (SELECT id, dept_name FROM departments),
emp_cte AS (SELECT * FROM employees)
SELECT e.name, d.dept_name FROM emp_cte e
JOIN dept_cte d ON e.dept_id = d.id;`
3. `WITH RECURSIVE emp_hierarchy AS (
SELECT id, name, manager_id FROM employees WHERE manager_id IS NULL
UNION ALL
SELECT e.id, e.name, e.manager_id
FROM employees e JOIN emp_hierarchy h ON e.manager_id = h.id
) SELECT * FROM emp_hierarchy;`



CTEs make complex queries easy to read and maintain.

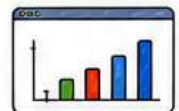


33.8 Window Function Problems

Problem	Description
1. ROW_NUMBER()	Assign row number in a partition.
2. RANK() / DENSE_RANK()	Rank rows within a partition.
3. LAG() / LEAD()	Access data from previous / next row.
4. SUM() OVER()	Running total using window function.
5. AVG() OVER()	Running average using window function.
6. NTILE()	Divide rows into N buckets.

Example Query :

1. `SELECT name, salary,
ROW_NUMBER() OVER (PARTITION BY dept_id ORDER BY salary DESC) rn
FROM employees;`
2. `SELECT name, salary,
RANK() OVER (PARTITION BY dept_id ORDER BY salary DESC) rnk
FROM employees;`
3. `SELECT name, salary,
LAG(salary) OVER (PARTITION BY dept_id ORDER BY salary) prev_salary,
LEAD(salary) OVER (PARTITION BY dept_id ORDER BY salary) next_salary
FROM employees;`
4. `SELECT name, dept_id, salary,
SUM(salary) OVER (PARTITION BY dept_id ORDER BY salary) running_total
FROM employees;`
5. `SELECT name, dept_id, salary,
AVG(salary) OVER (PARTITION BY dept_id) avg_salary
FROM employees;`
6. `SELECT name, salary,
NTILE(4) OVER (ORDER BY salary DESC) quartile
FROM employees;`



Window functions perform calculations across a set of rows related to the current row.



Key Takeaway :



Practice different types of problems to master MySQL.

Focus on **JOINS**, **Subqueries**, **CTEs** and **Window Functions** to solve real-world scenarios efficiently.



33.9 Optimization Problems

Problem	Description
1. Slow Queries	Identify slow queries and optimize them.
2. Index Usage	Create correct indexes to improve performance.
3. Unnecessary Columns	Avoid selecting unnecessary columns.
4. Large Tables	Optimize queries on large tables.
5. Join Optimization	Write efficient JOIN queries.
6. Subquery Optimization	Convert subqueries to JOINS where possible.

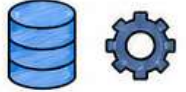
Key Takeaway :



Optimization improves query **performance**, reduces **execution time** and better utilizes **database resources**.

Optimization Tips :

1. Use indexes on columns used in WHERE, JOIN, ORDER BY and GROUP BY.
2. Avoid **SELECT ***, fetch only required columns.
3. Use **EXPLAIN** to analyze query execution plan.
4. Use **LIMIT** to restrict number of rows.
5. Avoid functions on indexed columns in **WHERE** clause.
6. Normalize database to reduce redundancy.
7. Use proper data types to save space and improve speed.
8. Use JOINS instead of subqueries for better performance.



Good database design + proper indexing = **high performance**

33.10 Real-World Scenario Questions

Scenario	Example Question
1. E-Commerce	Find top 5 best selling products.
2. Banking System	Find customers who did not make transactions in last 6 months.
3. HR System	List employees who earn more than the average salary.
4. Online Learning	Find students who scored more than 90% in all subjects.
5. Inventory System	Find products that are out of stock.
6. Hospital Management	List patients who have appointments today.

Example Query (E-Commerce) :

```
SELECT p.product_name, SUM(oi.quantity) AS total_sold
FROM products p
JOIN order_items oi ON p.product_id = oi.product_id
GROUP BY p.product_id, p.product_name
ORDER BY total_sold DESC
LIMIT 5;
```



More Scenarios :

- * **Social Media** - Find most active users.
- * **Sales System** - Monthly sales report.
- * **Online Booking** - Find most booked rooms.
- * **Payroll System** - Department wise salary report.
- * **Library System** - Books not issued in last 1 year.

Practice real-world scenarios to understand how MySQL solves business problems.



33.11 MySQL Coding Challenges

Challenge Type	Description
1. Basic Queries	Simple SELECT, WHERE, ORDER BY.
2. Aggregation	COUNT, SUM, AVG, GROUP BY.
3. Joins	INNER, LEFT, RIGHT, FULL JOIN.
4. Subqueries	IN, EXISTS, correlated subqueries.
5. CTEs	Use WITH clause for complex logic.
6. Window Functions	ROW_NUMBER(), RANK(), LAG(), LEAD().
7. Advanced Logic	CASE, IF, COALESCE, NULLIF.

Key Takeaway :



Coding challenges improve **problem-solving skills** and help in writing **efficient**, optimized and error-free queries.

Sample Challenge :

Find second highest salary from the employees table.

Query :

```
SELECT MAX(salary) AS second_highest
FROM employees
WHERE salary < (SELECT MAX(salary) FROM employees);
```



Other Challenges :

- * Find duplicate records.
- * Delete duplicate records keeping one.
- * Find nth highest salary.
- * Pivot data using conditional aggregation.
- * Write a query to find gaps in ID sequence.

33.12 MySQL Developer Interview Preparation

Preparation Areas
✓ Understand MySQL basics and architecture. ✓ Practice all types of queries (DDL, DML, DCL, TCL). ✓ Strong knowledge on JOINS, Subqueries, CTEs, Window Functions. ✓ Work on real-world projects. ✓ Solve query problems daily. ✓ Understand indexes, normalization and optimization. ✓ Review common errors and how to fix them. ✓ Practice explaining your queries clearly.

Interview Tips :

- * Read the question carefully.
- * Break the problem into smaller steps.
- * Write optimized and readable queries.
- * Test queries with edge cases.
- * Explain your thought process.
- * Be confident and communicate well.

Common Interview Rounds :

1. Technical MCQs
2. SQL Query Round
3. Coding Challenge
4. Real-World Scenario Round
5. System Design / DB Design
6. HR / Behavioral Round



Final Takeaway :



Consistent practice, strong fundamentals and real-world problem solving make you **job-ready** as a **MySQL Developer**.

