Sample Database: `company_db`

We’ll use this schema:

```
CREATE TABLE employees (
    emp_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    department VARCHAR(50),
    salary DECIMAL(10,2),
    date_hired DATE
);

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

CREATE TABLE projects (
    project_id INT AUTO_INCREMENT PRIMARY KEY,
    project_name VARCHAR(100),
    start_date DATE,
    end_date DATE
);

CREATE TABLE employee_projects (
    emp_id INT,
    project_id INT,
    role VARCHAR(50),
    PRIMARY KEY (emp_id, project_id),
    FOREIGN KEY (emp_id) REFERENCES employees(emp_id),
    FOREIGN KEY (project_id) REFERENCES projects(project_id)
);
```

Sample data:

```
INSERT INTO departments (dept_name, location) VALUES
('HR', 'New York'),
('IT', 'San Francisco'),
('Finance', 'Chicago');

INSERT INTO employees (name, department, salary, date_hired) VALUES
('Alice Smith', 'IT', 90000, '2020-01-15'),
('Bob Johnson', 'HR', 60000, '2019-03-10'),
('Carol Lee', 'Finance', 75000, '2021-07-01');

INSERT INTO projects (project_name, start_date, end_date) VALUES
('Project Apollo', '2021-01-01', '2021-12-31'),
('Project Zephyr', '2022-01-01', NULL);

INSERT INTO employee_projects (emp_id, project_id, role) VALUES
(1, 1, 'Developer'),
(1, 2, 'Lead Developer'),
(2, 1, 'Coordinator');
```

Easy (Level 1)

1. Write a query to list all employees.
2. Show all departments located in New York.
3. List projects that have no end date.
4. Export the `employees` table to a CSV file.
5. Import employee data from a CSV into `employees`.
6. Use `EXPLAIN` on a `SELECT * FROM employees`.
7. Show the process list of currently running queries.
8. Count the total number of employees.
9. Find employees hired after January 1, 2020.
10. Write a Node.js script to connect to MySQL and print the number of employees.
11. Write a Python script to fetch all departments.
12. Write a PHP script to display all projects in a table.
13. Use `mysqldump` to export `company_db`.
14. Restore `company_db` from a dump file.
15. Show the salary of each employee in ascending order.
16. Find employees working on Project Apollo.
17. Use `ANALYZE TABLE` on `employees`.
18. Show the current global status of connections.
19. Enable the slow query log and set threshold to 2 seconds.
20. Create an index on `department` in `employees`.

Intermediate (Level 2)

21. Write a query joining employees with departments to show employee name, department name, and location.
22. Export the `projects` table to JSON format.
23. Import a JSON file into `projects`.
24. Write a parameterized query in Python to get employees by department safely.
25. Demonstrate an unsafe query vulnerable to SQL injection.
26. Modify the unsafe query to prevent SQL injection.
27. Create a stored procedure to list all employees in a given department.
28. Create a trigger to log insertions into the `employees` table.
29. Show queries from the slow query log.
30. Create a backup of only the `projects` table using `mysqldump`.
31. Optimize the `employees` table.
32. Create a view showing employees and their current projects.
33. Write a Node.js script to insert a new department.
34. Use `SHOW ENGINE INNODB STATUS` to display lock information.
35. Create a transaction that inserts an employee and assigns them to a project.

36. Roll back a transaction if any insert fails.
37. Enable query profiling and measure execution time of selecting all employees.
38. Create a query that returns the highest salary.
39. List employees not assigned to any project.
40. List employees assigned to more than one project.

Advanced (Level 3)

41. Write a complex join returning employee names, their project names, and department locations.
42. Create a Node.js API endpoint that returns JSON of all employees.
43. Create a Python Flask API returning employees filtered by department.
44. Simulate concurrent inserts and demonstrate deadlock detection.
45. Design a backup strategy to export the entire DB nightly.
46. Configure MySQL to log all queries longer than 1 second.
47. Create a stored function that calculates an employee's tenure in years.
48. Create a generated column storing the year hired and index it.
49. Explain the performance impact of not using indexes on large tables.
50. Optimize a slow query with `EXPLAIN` and index recommendations.

Expert (Level 4)

51. Develop a Node.js app demonstrating SQL injection exploitation (in a safe environment) and mitigation.
52. Benchmark query performance with and without indexes on the `employee_projects` table.
53. Write a multi-step transaction inserting employees, assigning projects, and updating salaries atomically.
54. Configure automated backup rotation and retention for 30 days.
55. Design a solution to migrate the entire database to another server with minimal downtime.
56. Analyze and document the query execution plan of a complex 3-table join query.
57. Build a complete ETL pipeline to export employees to CSV, transform data (e.g., uppercase names), and import back into a new table.