**1. Create Table: departments**

CREATE TABLE departments (

department\_id INT PRIMARY KEY AUTO\_INCREMENT,

department\_name VARCHAR(100) NOT NULL

);

INSERT INTO departments (department\_name) VALUES

('Human Resources'),

('Finance'),

('IT'),

('Sales'),

('Marketing');

**2. Create Table: employees**

CREATE TABLE employees (

employee\_id INT PRIMARY KEY AUTO\_INCREMENT,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

salary DECIMAL(10,2) NOT NULL,

department\_id INT,

hire\_date DATE,

FOREIGN KEY (department\_id) REFERENCES departments(department\_id)

);

INSERT INTO employees (first\_name, last\_name, salary, department\_id, hire\_date) VALUES

('John', 'Doe', 55000, 1, '2020-06-15'),

('Jane', 'Smith', 75000, 2, '2018-03-10'),

('Michael', 'Brown', 45000, 3, '2019-11-25'),

('Emily', 'Davis', 62000, 4, '2021-07-19'),

('David', 'Wilson', 80000, 5, '2017-09-30');

**3. Create Table: managers**

CREATE TABLE managers (

manager\_id INT PRIMARY KEY AUTO\_INCREMENT,

manager\_name VARCHAR(100) NOT NULL,

department\_id INT,

FOREIGN KEY (department\_id) REFERENCES departments(department\_id)

);

INSERT INTO managers (manager\_name, department\_id) VALUES

('Alice Johnson', 1),

('Robert King', 2),

('Sophia White', 3),

('William Scott', 4),

('Linda Martinez', 5);

This setup ensures that:

1. The **departments** table contains department information.
2. The **employees** table stores employee details and references departments via department\_id.
3. The **managers** table keeps manager details, linking them to departments via department\_id.

**Lab 1: Basic SELECT Statement**

**Exercise:**  
Retrieve all columns from the employees table.

**Solution:**

SELECT \* FROM employees;

**Lab 2: Selecting Specific Columns**

**Exercise:**  
Retrieve only the first\_name and last\_name from the employees table.

**Solution:**

SELECT first\_name, last\_name FROM employees;

**Lab 3: Using WHERE Clause**

**Exercise:**  
Retrieve employees whose department\_id is 5.

**Solution:**

SELECT \* FROM employees WHERE department\_id = 5;

**Lab 4: Using Comparison Operators**

**Exercise:**  
Retrieve employees whose salary is greater than 50,000.

**Solution:**

SELECT \* FROM employees WHERE salary > 50000;

**Lab 5: Using Logical Operators**

**Exercise:**  
Retrieve employees who work in department\_id 3 or 4 and earn more than 40,000.

**Solution:**

SELECT \* FROM employees WHERE (department\_id = 3 OR department\_id = 4) AND salary > 40000;

**Lab 6: Using BETWEEN Operator**

**Exercise:**  
Retrieve employees whose salary is between 30,000 and 60,000.

**Solution:**

SELECT \* FROM employees WHERE salary BETWEEN 30000 AND 60000;

**Lab 7: Using IN Operator**

**Exercise:**  
Retrieve employees who work in departments 2, 3, or 5.

**Solution:**

SELECT \* FROM employees WHERE department\_id IN (2, 3, 5);

**Lab 8: Using LIKE Operator**

**Exercise:**  
Retrieve employees whose first name starts with "J".

**Solution:**

SELECT \* FROM employees WHERE first\_name LIKE 'J%';

**Lab 9: Using ORDER BY Clause**

**Exercise:**  
Retrieve employees sorted by salary in descending order.

**Solution:**

SELECT \* FROM employees ORDER BY salary DESC;

**Lab 10: Using LIMIT Clause**

**Exercise:**  
Retrieve the top 5 highest-paid employees.

**Solution:**

SELECT \* FROM employees ORDER BY salary DESC LIMIT 5;

**Lab 11: Using Aggregate Functions (COUNT)**

**Exercise:**  
Find the total number of employees in the company.

**Solution:**

SELECT COUNT(\*) AS total\_employees FROM employees;

**Lab 12: Using Aggregate Functions (SUM)**

**Exercise:**  
Find the total salary expense for all employees.

**Solution:**

SELECT SUM(salary) AS total\_salary\_expense FROM employees;

**Lab 13: Using Aggregate Functions (AVG)**

**Exercise:**  
Find the average salary of employees.

**Solution:**

SELECT AVG(salary) AS average\_salary FROM employees;

**Lab 14: Using GROUP BY Clause**

**Exercise:**  
Find the total number of employees in each department.

**Solution:**

SELECT department\_id, COUNT(\*) AS employee\_count FROM employees GROUP BY department\_id;

**Lab 15: Using GROUP BY with HAVING Clause**

**Exercise:**  
Find departments that have more than 10 employees.

**Solution:**

SELECT department\_id, COUNT(\*) AS employee\_count

FROM employees

GROUP BY department\_id

HAVING employee\_count > 10;

**Lab 16: Using INNER JOIN**

**Exercise:**  
Retrieve employee names along with their department names. (Assume departments table has department\_id and department\_name.)

**Solution:**

SELECT e.first\_name, e.last\_name, d.department\_name

FROM employees e

INNER JOIN departments d ON e.department\_id = d.department\_id;

**Lab 17: Using LEFT JOIN**

**Exercise:**  
Retrieve all employees and their department names, including those who don't belong to any department.

**Solution:**

SELECT e.first\_name, e.last\_name, d.department\_name

FROM employees e

LEFT JOIN departments d ON e.department\_id = d.department\_id;

**Lab 18: Using Subquery in WHERE Clause**

**Exercise:**  
Retrieve employees who earn more than the average salary.

**Solution:**

SELECT \* FROM employees

WHERE salary > (SELECT AVG(salary) FROM employees);

**Lab 19: Using EXISTS with Subquery**

**Exercise:**  
Retrieve employees who work in departments that have at least one manager. (Assume managers table contains department\_id.)

**Solution:**

SELECT \* FROM employees e

WHERE EXISTS (SELECT 1 FROM managers m WHERE e.department\_id = m.department\_id);

**Lab 20: Using CASE Statement**

**Exercise:**  
Display employee salary grades:

* "High" if salary > 60,000
* "Medium" if salary between 30,000 and 60,000
* "Low" otherwise

**Solution:**

SELECT first\_name, last\_name, salary,

CASE

WHEN salary > 60000 THEN 'High'

WHEN salary BETWEEN 30000 AND 60000 THEN 'Medium'

ELSE 'Low'

END AS salary\_grade

FROM employees;

**UPDATE Statement Exercises**

**Lab 1: Basic UPDATE**

**Exercise:**  
Increase the salary of all employees in department 3 by 10%.

**Solution:**

UPDATE employees

SET salary = salary \* 1.10

WHERE department\_id = 3;

**Lab 2: Updating a Single Record**

**Exercise:**  
Change the last name of employee employee\_id = 2 to "Johnson".

**Solution:**

UPDATE employees

SET last\_name = 'Johnson'

WHERE employee\_id = 2;

**Lab 3: Updating Multiple Columns**

**Exercise:**  
Change salary to 70000 and department\_id to 2 for employee employee\_id = 3.

**Solution:**

UPDATE employees

SET salary = 70000, department\_id = 2

WHERE employee\_id = 3;

**Lab 4: Conditional UPDATE**

**Exercise:**  
Set the department of employees earning less than 50000 to department 1.

**Solution:**

UPDATE employees

SET department\_id = 1

WHERE salary < 50000;

**Lab 5: Using UPDATE with a Subquery**

**Exercise:**  
Increase the salary of employees in the highest-paid department by 15%.

**Solution:**

UPDATE employees

SET salary = salary \* 1.15

WHERE department\_id = (SELECT department\_id FROM employees ORDER BY salary DESC LIMIT 1);

**Lab 6: Updating Using JOIN**

**Exercise:**  
Set all employees in the Sales department to have a 70000 salary.

**Solution:**

UPDATE employees e

JOIN departments d ON e.department\_id = d.department\_id

SET e.salary = 70000

WHERE d.department\_name = 'Sales';

**Lab 7: Using UPDATE with LIKE**

**Exercise:**  
Change the department of employees whose names start with 'J' to Marketing.

**Solution:**

UPDATE employees

SET department\_id = (SELECT department\_id FROM departments WHERE department\_name = 'Marketing')

WHERE first\_name LIKE 'J%';

**Lab 8: Updating Based on Tenure**

**Exercise:**  
Increase the salary by 20% for employees hired before 2020.

**Solution:**

UPDATE employees

SET salary = salary \* 1.20

WHERE hire\_date < '2020-01-01';

**Lab 9: Using CASE in UPDATE**

**Exercise:**  
Set employee salary based on performance levels:

* High performers get a 20% raise
* Medium performers get a 10% raise
* Low performers get no raise

**Solution:**

UPDATE employees

SET salary = CASE

WHEN salary > 70000 THEN salary \* 1.20

WHEN salary BETWEEN 40000 AND 70000 THEN salary \* 1.10

ELSE salary

END;

**Lab 10: Resetting Column Values**

**Exercise:**  
Set all salaries to 50000 for employees in IT department.

**Solution:**

UPDATE employees

SET salary = 50000

WHERE department\_id = (SELECT department\_id FROM departments WHERE department\_name = 'IT');

**DELETE Statement Exercises**

**Lab 11: Basic DELETE**

**Exercise:**  
Delete an employee with employee\_id = 4.

**Solution:**

DELETE FROM employees

WHERE employee\_id = 4;

**Lab 12: Deleting Multiple Rows**

**Exercise:**  
Delete all employees in the Marketing department.

**Solution:**

DELETE FROM employees

WHERE department\_id = (SELECT department\_id FROM departments WHERE department\_name = 'Marketing');

**Lab 13: Using DELETE with a Condition**

**Exercise:**  
Delete employees earning less than 35000.

**Solution:**

DELETE FROM employees

WHERE salary < 35000;

**Lab 14: Using DELETE with LIMIT**

**Exercise:**  
Delete only the first 2 employees with the lowest salary.

**Solution:**

DELETE FROM employees

ORDER BY salary ASC

LIMIT 2;

**Lab 15: DELETE Using JOIN**

**Exercise:**  
Delete all employees from departments that do not have managers.

**Solution:**

DELETE FROM employees e

LEFT JOIN managers m ON e.department\_id = m.department\_id

WHERE m.manager\_id IS NULL;

**Lab 16: DELETE with Subquery**

**Exercise:**  
Delete employees hired before 2015.

**Solution:**

DELETE FROM employees

WHERE hire\_date < '2015-01-01';

**Lab 17: Using DELETE with NOT IN**

**Exercise:**  
Delete employees who are not in Finance or HR departments.

**Solution:**

DELETE FROM employees

WHERE department\_id NOT IN (

SELECT department\_id FROM departments WHERE department\_name IN ('Finance', 'Human Resources')

);

**Lab 18: Using DELETE with EXISTS**

**Exercise:**  
Delete employees who work in departments where no employees earn above 60000.

**Solution:**

DELETE FROM employees e

WHERE NOT EXISTS (

SELECT 1 FROM employees e2 WHERE e2.department\_id = e.department\_id AND e2.salary > 60000

);

**Lab 19: DELETE Employees with Duplicate Names**

**Exercise:**  
Delete employees with duplicate first and last names, keeping the one with the highest salary.

**Solution:**

DELETE e1 FROM employees e1

JOIN employees e2

ON e1.first\_name = e2.first\_name AND e1.last\_name = e2.last\_name

WHERE e1.salary < e2.salary;

**Lab 20: Deleting All Employees from a Department (Cascading)**

**Exercise:**  
Delete all employees from the Sales department and then remove the department.

**Solution:**

DELETE FROM employees

WHERE department\_id = (SELECT department\_id FROM departments WHERE department\_name = 'Sales');

DELETE FROM departments

WHERE department\_name = 'Sales';

**1. ALTER - Add a New Column**

**Exercise:**  
Add a new column email VARCHAR(100) to the employees table.

**Solution:**

ALTER TABLE employees

ADD COLUMN email VARCHAR(100);

**2. ALTER - Add Multiple Columns**

**Exercise:**  
Add phone\_number VARCHAR(15) and date\_of\_birth DATE to the employees table.

**Solution:**

ALTER TABLE employees

ADD COLUMN phone\_number VARCHAR(15),

ADD COLUMN date\_of\_birth DATE;

**3. ALTER - Modify Column Data Type**

**Exercise:**  
Change the salary column in employees to DECIMAL(12,2).

**Solution:**

ALTER TABLE employees

MODIFY COLUMN salary DECIMAL(12,2);

**4. ALTER - Rename Column**

**Exercise:**  
Rename the column phone\_number to contact\_number in employees.

**Solution:**

ALTER TABLE employees

CHANGE COLUMN phone\_number contact\_number VARCHAR(20);

**5. ALTER - Drop a Column**

**Exercise:**  
Remove the email column from employees.

**Solution:**

ALTER TABLE employees

DROP COLUMN email;

**6. ALTER - Add a Primary Key**

**Exercise:**  
Add a primary key to the departments table on department\_id.

**Solution:**

ALTER TABLE departments

ADD PRIMARY KEY (department\_id);

**7. ALTER - Add a Unique Constraint**

**Exercise:**  
Ensure email is unique in employees.

**Solution:**

ALTER TABLE employees

ADD CONSTRAINT unique\_email UNIQUE (email);

**8. ALTER - Add a Foreign Key**

**Exercise:**  
Add a foreign key constraint from employees.department\_id to departments.department\_id.

**Solution:**

ALTER TABLE employees

ADD CONSTRAINT fk\_department

FOREIGN KEY (department\_id) REFERENCES departments(department\_id);

**9. ALTER - Drop a Foreign Key**

**Exercise:**  
Remove the foreign key fk\_department from employees.

**Solution:**

ALTER TABLE employees

DROP FOREIGN KEY fk\_department;

**10. ALTER - Drop a Primary Key**

**Exercise:**  
Remove the primary key from departments.

**Solution:**

ALTER TABLE departments

DROP PRIMARY KEY;

**11. ALTER - Set Default Value**

**Exercise:**  
Set a default salary of 50000 for employees.

**Solution:**

ALTER TABLE employees

ALTER COLUMN salary SET DEFAULT 50000;

**12. ALTER - Drop Default Value**

**Exercise:**  
Remove the default value from salary.

**Solution:**

ALTER TABLE employees

ALTER COLUMN salary DROP DEFAULT;

**13. ALTER - Change Column Order**

**Exercise:**  
Move the hire\_date column to be after last\_name.

**Solution:**

ALTER TABLE employees

MODIFY COLUMN hire\_date DATE AFTER last\_name;

**14. ALTER - Rename Table**

**Exercise:**  
Rename the employees table to staff\_members.

**Solution:**

ALTER TABLE employees

RENAME TO staff\_members;

**15. ALTER - Add an Index**

**Exercise:**  
Create an index on last\_name in employees.

**Solution:**

ALTER TABLE employees

ADD INDEX idx\_lastname (last\_name);

**16. ALTER - Drop an Index**

**Exercise:**  
Remove the index idx\_lastname from employees.

**Solution:**

ALTER TABLE employees

DROP INDEX idx\_lastname;

**17. ALTER - Add a Composite Index**

**Exercise:**  
Create an index on first\_name and last\_name for faster searches.

**Solution:**

ALTER TABLE employees

ADD INDEX idx\_name (first\_name, last\_name);

**18. ALTER - Convert a Column to NOT NULL**

**Exercise:**  
Ensure last\_name cannot be NULL in employees.

**Solution:**

ALTER TABLE employees

MODIFY COLUMN last\_name VARCHAR(50) NOT NULL;

**19. ALTER - Remove NOT NULL Constraint**

**Exercise:**  
Allow last\_name to be NULL in employees.

**Solution:**

ALTER TABLE employees

MODIFY COLUMN last\_name VARCHAR(50) NULL;

**20. ALTER - Add an AUTO\_INCREMENT**

**Exercise:**  
Set employee\_id as an auto-increment primary key.

**Solution:**

ALTER TABLE employees

MODIFY COLUMN employee\_id INT AUTO\_INCREMENT PRIMARY KEY;

**Partitioning in MySQL**

**1. Creating a Partitioned Table by Range**

**Exercise:**  
Partition the employees table based on salary into 3 ranges:

* 0 - 50,000
* 50,001 - 100,000
* 100,001 and above

**Solution:**

CREATE TABLE employees\_partitioned (

employee\_id INT NOT NULL,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

salary DECIMAL(10,2),

department\_id INT,

hire\_date DATE,

PRIMARY KEY (employee\_id, salary)

)

PARTITION BY RANGE (salary) (

PARTITION p1 VALUES LESS THAN (50001),

PARTITION p2 VALUES LESS THAN (100001),

PARTITION p3 VALUES LESS THAN MAXVALUE

);

**2. Creating a Partitioned Table by List**

**Exercise:**  
Partition employees based on department\_id. Departments 1,2,3 go into one partition, 4,5,6 into another.

**Solution:**

CREATE TABLE employees\_partitioned\_list (

employee\_id INT NOT NULL,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

salary DECIMAL(10,2),

department\_id INT NOT NULL,

PRIMARY KEY (employee\_id, department\_id)

)

PARTITION BY LIST (department\_id) (

PARTITION p\_hr\_finance VALUES IN (1,2,3),

PARTITION p\_it\_sales VALUES IN (4,5,6)

);

**3. Partitioning by Hash**

**Exercise:**  
Partition the employees table into 4 partitions using a **hash partitioning** on department\_id.

**Solution:**

CREATE TABLE employees\_partitioned\_hash (

employee\_id INT NOT NULL,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

salary DECIMAL(10,2),

department\_id INT,

PRIMARY KEY (employee\_id, department\_id)

)

PARTITION BY HASH (department\_id)

PARTITIONS 4;

**4. Partitioning by Key**

**Exercise:**  
Partition the employees table using **key partitioning** on employee\_id into 5 partitions.

**Solution:**

CREATE TABLE employees\_partitioned\_key (

employee\_id INT NOT NULL,

first\_name VARCHAR(50),

last\_name VARCHAR(50),

salary DECIMAL(10,2),

department\_id INT,

PRIMARY KEY (employee\_id)

)

PARTITION BY KEY(employee\_id)

PARTITIONS 5;

**5. Dropping a Partition**

**Exercise:**  
Drop the p2 partition from employees\_partitioned.

**Solution:**

ALTER TABLE employees\_partitioned DROP PARTITION p2;

**Triggers in MySQL**

**6. Creating a BEFORE INSERT Trigger**

**Exercise:**  
Prevent employees from having a salary below 30,000 using a **BEFORE INSERT** trigger.

**Solution:**

DELIMITER $$

CREATE TRIGGER before\_employee\_insert

BEFORE INSERT ON employees

FOR EACH ROW

BEGIN

IF NEW.salary < 30000 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Salary cannot be below 30,000!';

END IF;

END $$

DELIMITER ;

**7. Creating an AFTER INSERT Trigger**

**Exercise:**  
Log new employees into a employee\_audit table.

**Solution:**

CREATE TABLE employee\_audit (

audit\_id INT AUTO\_INCREMENT PRIMARY KEY,

employee\_id INT,

action VARCHAR(50),

timestamp DATETIME DEFAULT CURRENT\_TIMESTAMP

);

DELIMITER $$

CREATE TRIGGER after\_employee\_insert

AFTER INSERT ON employees

FOR EACH ROW

BEGIN

INSERT INTO employee\_audit (employee\_id, action)

VALUES (NEW.employee\_id, 'INSERTED');

END $$

DELIMITER ;

**8. Creating a BEFORE UPDATE Trigger**

**Exercise:**  
Prevent salary decreases in the employees table.

**Solution:**

DELIMITER $$

CREATE TRIGGER before\_employee\_update

BEFORE UPDATE ON employees

FOR EACH ROW

BEGIN

IF NEW.salary < OLD.salary THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Salary cannot be decreased!';

END IF;

END $$

DELIMITER ;

**9. Creating an AFTER UPDATE Trigger**

**Exercise:**  
Log salary changes in employee\_salary\_audit.

**Solution:**

CREATE TABLE employee\_salary\_audit (

audit\_id INT AUTO\_INCREMENT PRIMARY KEY,

employee\_id INT,

old\_salary DECIMAL(10,2),

new\_salary DECIMAL(10,2),

timestamp DATETIME DEFAULT CURRENT\_TIMESTAMP

);

DELIMITER $$

CREATE TRIGGER after\_employee\_salary\_update

AFTER UPDATE ON employees

FOR EACH ROW

BEGIN

IF OLD.salary <> NEW.salary THEN

INSERT INTO employee\_salary\_audit (employee\_id, old\_salary, new\_salary)

VALUES (NEW.employee\_id, OLD.salary, NEW.salary);

END IF;

END $$

DELIMITER ;

**10. Creating a BEFORE DELETE Trigger**

**Exercise:**  
Prevent deletion of employees with a salary above 100,000.

**Solution:**

DELIMITER $$

CREATE TRIGGER before\_employee\_delete

BEFORE DELETE ON employees

FOR EACH ROW

BEGIN

IF OLD.salary > 100000 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Cannot delete high-salary employees!';

END IF;

END $$

DELIMITER ;

**11. Creating an AFTER DELETE Trigger**

**Exercise:**  
Log deletions in employee\_deletion\_log.

**Solution:**

CREATE TABLE employee\_deletion\_log (

log\_id INT AUTO\_INCREMENT PRIMARY KEY,

employee\_id INT,

deleted\_at DATETIME DEFAULT CURRENT\_TIMESTAMP

);

DELIMITER $$

CREATE TRIGGER after\_employee\_delete

AFTER DELETE ON employees

FOR EACH ROW

BEGIN

INSERT INTO employee\_deletion\_log (employee\_id)

VALUES (OLD.employee\_id);

END $$

DELIMITER ;

**12. Using Triggers to Auto-Calculate Years of Service**

**Exercise:**  
Automatically update years\_of\_service column when a new employee is inserted.

**Solution:**

ALTER TABLE employees ADD COLUMN years\_of\_service INT;

DELIMITER $$

CREATE TRIGGER after\_employee\_insert\_service

AFTER INSERT ON employees

FOR EACH ROW

BEGIN

UPDATE employees

SET years\_of\_service = TIMESTAMPDIFF(YEAR, NEW.hire\_date, CURDATE())

WHERE employee\_id = NEW.employee\_id;

END $$

DELIMITER ;

**13. Using Triggers for Data Validation**

**Exercise:**  
Ensure that employees can only be assigned to existing department\_id values.

**Solution:**

DELIMITER $$

CREATE TRIGGER before\_employee\_insert\_validate\_department

BEFORE INSERT ON employees

FOR EACH ROW

BEGIN

IF (SELECT COUNT(\*) FROM departments WHERE department\_id = NEW.department\_id) = 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Invalid department ID!';

END IF;

END $$

DELIMITER ;