**Step 1: Database & Tables**

**Database Creation**

CREATE DATABASE CompanyDB;

USE CompanyDB;

**Table: Employees**

CREATE TABLE Employees (

emp\_id INT PRIMARY KEY,

emp\_name VARCHAR(50),

department VARCHAR(50),

salary DECIMAL(10, 2)

);

**Insert Sample Data:**

INSERT INTO Employees VALUES

(1, 'Alice', 'HR', 5000),

(2, 'Bob', 'IT', 6000),

(3, 'Charlie', 'Finance', 5500),

(4, 'David', 'IT', 7000),

(5, 'Eva', 'HR', 4800);

**Table: Audit\_Log (for Trigger Logging)**

CREATE TABLE Audit\_Log (

log\_id INT AUTO\_INCREMENT PRIMARY KEY,

emp\_id INT,

action VARCHAR(20),

action\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

**Exercise 1: Log Employee Insertions**

**Task:** Create a trigger to log every new employee added to the Employees table.

**Solution:**

DELIMITER $$

CREATE TRIGGER after\_employee\_insert

AFTER INSERT ON Employees

FOR EACH ROW

BEGIN

INSERT INTO Audit\_Log (emp\_id, action)

VALUES (NEW.emp\_id, 'INSERT');

END$$

DELIMITER ;

**Test It:**

INSERT INTO Employees VALUES (6, 'Frank', 'Marketing', 5200);

SELECT \* FROM Audit\_Log;

**Expected Output:**

| **log\_id** | **emp\_id** | **action** | **action\_time** |
| --- | --- | --- | --- |
| 1 | 6 | INSERT | *timestamp* |

**Exercise 2: Track Salary Updates**

**Task:** Log whenever an employee's salary is updated.

**Solution:**

DELIMITER $$

CREATE TRIGGER after\_salary\_update

AFTER UPDATE ON Employees

FOR EACH ROW

BEGIN

IF OLD.salary <> NEW.salary THEN

INSERT INTO Audit\_Log (emp\_id, action)

VALUES (NEW.emp\_id, 'SALARY UPDATE');

END IF;

END$$

DELIMITER ;

**Test It:**

UPDATE Employees SET salary = 6200 WHERE emp\_id = 2;

SELECT \* FROM Audit\_Log;

**Expected Output:**

| **log\_id** | **emp\_id** | **action** | **action\_time** |
| --- | --- | --- | --- |
| 1 | 6 | INSERT | *timestamp* |
| 2 | 2 | SALARY UPDATE | *timestamp* |

**Exercise 3: Prevent Deletion of High-Salary Employees**

**Task:** Prevent deletion of employees with a salary above 6000.

DELIMITER $$

CREATE TRIGGER prevent\_high\_salary\_delete

BEFORE DELETE ON Employees

FOR EACH ROW

BEGIN

IF OLD.salary > 6000 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Cannot delete employees with salary above 6000!';

END IF;

END$$

DELIMITER ;

**Test It:**

DELETE FROM Employees WHERE emp\_id = 4; -- David's salary is 7000

**Expected Output:**

ERROR 1644 (45000): Cannot delete employees with salary above 6000!

**Exercise 4: Auto-Adjust Salary for New Employees**

**Task:** Automatically increase the salary of new employees by 5% if they belong to the IT department.

**Solution:**

DELIMITER $$

CREATE TRIGGER auto\_adjust\_it\_salary

BEFORE INSERT ON Employees

FOR EACH ROW

BEGIN

IF NEW.department = 'IT' THEN

SET NEW.salary = NEW.salary \* 1.05;

END IF;

END$$

DELIMITER ;

**Test It:**

INSERT INTO Employees VALUES (7, 'Grace', 'IT', 6500);

SELECT \* FROM Employees WHERE emp\_id = 7;

**Expected Output:**

| **emp\_id** | **emp\_name** | **department** | **salary** |
| --- | --- | --- | --- |
| 7 | Grace | IT | 6825.00 |

**Exercise 5: Maintain Department Headcount**

**Task:** Create a Department\_Stats table to track the number of employees in each department.

**Step 1: Create the Table**

CREATE TABLE Department\_Stats (

department VARCHAR(50) PRIMARY KEY,

headcount INT

);

**Step 2: Insert Initial Headcounts**

INSERT INTO Department\_Stats (department, headcount)

SELECT department, COUNT(\*) FROM Employees GROUP BY department;

**Trigger for Insertion:**

DELIMITER $$

CREATE TRIGGER after\_employee\_insert\_update\_count

AFTER INSERT ON Employees

FOR EACH ROW

BEGIN

INSERT INTO Department\_Stats (department, headcount)

VALUES (NEW.department, 1)

ON DUPLICATE KEY UPDATE headcount = headcount + 1;

END$$

DELIMITER ;

**Trigger for Deletion:**

DELIMITER $$

CREATE TRIGGER after\_employee\_delete\_update\_count

AFTER DELETE ON Employees

FOR EACH ROW

BEGIN

UPDATE Department\_Stats

SET headcount = headcount - 1

WHERE department = OLD.department;

END$$

DELIMITER ;

**Test It:**

INSERT INTO Employees VALUES (8, 'Helen', 'Finance', 5300);

DELETE FROM Employees WHERE emp\_id = 3; -- Charlie from Finance

SELECT \* FROM Department\_Stats;

**Expected Output:**

| **department** | **headcount** |
| --- | --- |
| HR | 2 |
| IT | 3 |
| Finance | 1 |
| Marketing | 1 |

**Exercise 6: Auto-Generate Employee ID**

**Task:** Create a trigger to automatically generate a unique emp\_id for new employees if not provided.

**Solution:**

DELIMITER $$

CREATE TRIGGER auto\_generate\_emp\_id

BEFORE INSERT ON Employees

FOR EACH ROW

BEGIN

IF NEW.emp\_id IS NULL THEN

SET NEW.emp\_id = (SELECT IFNULL(MAX(emp\_id), 0) + 1 FROM Employees);

END IF;

END$$

DELIMITER ;

**Test It:**

INSERT INTO Employees (emp\_name, department, salary) VALUES ('Irene', 'HR', 5100);

SELECT \* FROM Employees WHERE emp\_name = 'Irene';

**Expected Output:**

| **emp\_id** | **emp\_name** | **department** | **salary** |
| --- | --- | --- | --- |
| 9 | Irene | HR | 5100 |

**Exercise 7: Enforce Salary Cap**

**Task:** Prevent salary updates exceeding $10,000 for any employee.

**Solution:**

DELIMITER $$

CREATE TRIGGER prevent\_high\_salary\_update

BEFORE UPDATE ON Employees

FOR EACH ROW

BEGIN

IF NEW.salary > 10000 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Salary cannot exceed $10,000!';

END IF;

END$$

DELIMITER ;

**Test It:**

UPDATE Employees SET salary = 15000 WHERE emp\_id = 1; -- Alice

**Expected Output:**

ERROR 1644 (45000): Salary cannot exceed $10,000!

**Exercise 8: Bonus Calculation on Salary Update**

**Task:** Automatically add a 10% bonus whenever an employee’s salary is increased.

**Solution:**

DELIMITER $$

CREATE TRIGGER add\_bonus\_after\_salary\_increase

AFTER UPDATE ON Employees

FOR EACH ROW

BEGIN

IF NEW.salary > OLD.salary THEN

UPDATE Employees

SET salary = NEW.salary \* 1.10

WHERE emp\_id = NEW.emp\_id;

END IF;

END$$

DELIMITER ;

**Test It:**

UPDATE Employees SET salary = 6200 WHERE emp\_id = 5; -- Eva

SELECT \* FROM Employees WHERE emp\_id = 5;

**Expected Output:**

| **emp\_id** | **emp\_name** | **department** | **salary** |
| --- | --- | --- | --- |
| 5 | Eva | HR | 6820 |

**Exercise 9: Maintain Employee History**

**Task:** Track all changes made to employee details (name, department, or salary) in a new Employee\_History table.

**Step 1: Create the History Table**

CREATE TABLE Employee\_History (

history\_id INT AUTO\_INCREMENT PRIMARY KEY,

emp\_id INT,

field\_changed VARCHAR(50),

old\_value VARCHAR(100),

new\_value VARCHAR(100),

change\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

**Trigger to Track Changes:**

DELIMITER $$

CREATE TRIGGER track\_employee\_changes

AFTER UPDATE ON Employees

FOR EACH ROW

BEGIN

IF OLD.emp\_name <> NEW.emp\_name THEN

INSERT INTO Employee\_History (emp\_id, field\_changed, old\_value, new\_value)

VALUES (NEW.emp\_id, 'Name', OLD.emp\_name, NEW.emp\_name);

END IF;

IF OLD.department <> NEW.department THEN

INSERT INTO Employee\_History (emp\_id, field\_changed, old\_value, new\_value)

VALUES (NEW.emp\_id, 'Department', OLD.department, NEW.department);

END IF;

IF OLD.salary <> NEW.salary THEN

INSERT INTO Employee\_History (emp\_id, field\_changed, old\_value, new\_value)

VALUES (NEW.emp\_id, 'Salary', OLD.salary, NEW.salary);

END IF;

END$$

DELIMITER ;

**Test It:**

UPDATE Employees SET emp\_name = 'Eva Smith', department = 'Finance', salary = 7200 WHERE emp\_id = 5;

SELECT \* FROM Employee\_History WHERE emp\_id = 5;

**Expected Output:**

| **history\_id** | **emp\_id** | **field\_changed** | **old\_value** | **new\_value** | **change\_date** |
| --- | --- | --- | --- | --- | --- |
| 1 | 5 | Name | Eva | Eva Smith | *timestamp* |
| 2 | 5 | Department | HR | Finance | *timestamp* |
| 3 | 5 | Salary | 6820 | 7200 | *timestamp* |

**Exercise 10: Prevent Deletion of Last Employee in a Department**

**Task:** Ensure that at least one employee remains in every department. Prevent deleting the last employee.

**Solution:**

DELIMITER $$

CREATE TRIGGER prevent\_last\_employee\_delete

BEFORE DELETE ON Employees

FOR EACH ROW

BEGIN

IF (SELECT COUNT(\*) FROM Employees WHERE department = OLD.department) = 1 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Cannot delete the last employee in the department!';

END IF;

END$$

DELIMITER ;

**Test It:**

DELETE FROM Employees WHERE emp\_id = 8; -- Helen from Finance (if she's the last)

**Expected Output:**

ERROR 1644 (45000): Cannot delete the last employee in the department!

**Exercise 11: Track Failed Login Attempts**

**Task:** Assume there's a Users table. Create a trigger to log failed login attempts in a Login\_Attempts table.

**Step 1: Create Users and Login\_Attempts Tables**

CREATE TABLE Users (

user\_id INT PRIMARY KEY,

username VARCHAR(50),

password VARCHAR(50) -- For simplicity (not recommended in real-world)

);

CREATE TABLE Login\_Attempts (

attempt\_id INT AUTO\_INCREMENT PRIMARY KEY,

user\_id INT,

attempt\_status VARCHAR(20),

attempt\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

**Insert Sample Data:**

INSERT INTO Users VALUES (1, 'john\_doe', 'pass123'), (2, 'jane\_smith', 'secret');

**Trigger for Failed Attempts:**

DELIMITER $$

CREATE TRIGGER log\_failed\_login

AFTER INSERT ON Login\_Attempts

FOR EACH ROW

BEGIN

IF NEW.attempt\_status = 'FAILED' THEN

INSERT INTO Login\_Attempts (user\_id, attempt\_status)

VALUES (NEW.user\_id, 'FAILED');

END IF;

END$$

DELIMITER ;

**Test It:**

INSERT INTO Login\_Attempts (user\_id, attempt\_status) VALUES (1, 'FAILED');

SELECT \* FROM Login\_Attempts;

**Expected Output:**

| **attempt\_id** | **user\_id** | **attempt\_status** | **attempt\_time** |
| --- | --- | --- | --- |
| 1 | 1 | FAILED | *timestamp* |

**Exercise 12: Log Deleted Employee Information**

**Task:** Create a trigger to log details of employees when they are deleted.

**Step 1: Create the Log Table**

CREATE TABLE Deleted\_Employees\_Log (

log\_id INT AUTO\_INCREMENT PRIMARY KEY,

emp\_id INT,

emp\_name VARCHAR(50),

department VARCHAR(50),

salary DECIMAL(10,2),

deleted\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

**Trigger to Log Deletions:**

DELIMITER $$

CREATE TRIGGER log\_employee\_deletion

AFTER DELETE ON Employees

FOR EACH ROW

BEGIN

INSERT INTO Deleted\_Employees\_Log (emp\_id, emp\_name, department, salary)

VALUES (OLD.emp\_id, OLD.emp\_name, OLD.department, OLD.salary);

END$$

DELIMITER ;

**Test It:**

DELETE FROM Employees WHERE emp\_id = 4; -- Deleting David

SELECT \* FROM Deleted\_Employees\_Log;

**Expected Output:**

| **log\_id** | **emp\_id** | **emp\_name** | **department** | **salary** | **deleted\_at** |
| --- | --- | --- | --- | --- | --- |
| 1 | 4 | David | IT | 7000 | *timestamp* |

**Exercise 13: Restrict Inserts on Weekends**

**Task:** Prevent new employee records from being added on weekends.

**Trigger to Restrict Inserts:**

DELIMITER $$

CREATE TRIGGER prevent\_weekend\_inserts

BEFORE INSERT ON Employees

FOR EACH ROW

BEGIN

IF DAYOFWEEK(CURRENT\_DATE()) IN (1, 7) THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Cannot insert employee records on weekends!';

END IF;

END$$

DELIMITER ;

**Test It:**

INSERT INTO Employees (emp\_name, department, salary) VALUES ('John', 'Marketing', 5500);

**Expected Output (if run on Saturday/Sunday):**

ERROR 1644 (45000): Cannot insert employee records on weekends!

**Exercise 14: Prevent Salary Reduction for Managers**

**Task:** Ensure that a manager’s salary cannot be reduced during an update.

**Trigger to Prevent Salary Reduction:**

DELIMITER $$

CREATE TRIGGER prevent\_manager\_salary\_cut

BEFORE UPDATE ON Employees

FOR EACH ROW

BEGIN

IF NEW.department = 'Management' AND NEW.salary < OLD.salary THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Cannot reduce the salary of a manager!';

END IF;

END$$

DELIMITER ;

**Test It:**

UPDATE Employees SET salary = 8000 WHERE emp\_name = 'Alice'; -- Assuming Alice is in 'Management'

**Expected Output:**

ERROR 1644 (45000): Cannot reduce the salary of a manager!

**Exercise 15: Update Department Employee Count Automatically**

**Task:** Keep track of the total number of employees in each department after insertions and deletions.

**Step 1: Create the Department Summary Table**

CREATE TABLE Department\_Summary (

department VARCHAR(50) PRIMARY KEY,

employee\_count INT

);

**Insert Initial Data:**

INSERT INTO Department\_Summary (department, employee\_count)

SELECT department, COUNT(\*) FROM Employees GROUP BY department;

**Triggers to Maintain Employee Count:**

**Trigger for Insertions:**

DELIMITER $$

CREATE TRIGGER increment\_employee\_count

AFTER INSERT ON Employees

FOR EACH ROW

BEGIN

INSERT INTO Department\_Summary (department, employee\_count)

VALUES (NEW.department, 1)

ON DUPLICATE KEY UPDATE employee\_count = employee\_count + 1;

END$$

DELIMITER ;

**Trigger for Deletions:**

DELIMITER $$

CREATE TRIGGER decrement\_employee\_count

AFTER DELETE ON Employees

FOR EACH ROW

BEGIN

UPDATE Department\_Summary

SET employee\_count = employee\_count - 1

WHERE department = OLD.department;

END$$

DELIMITER ;

**Test It:**

INSERT INTO Employees (emp\_name, department, salary) VALUES ('Chris', 'HR', 4500);

DELETE FROM Employees WHERE emp\_name = 'Eva Smith';

SELECT \* FROM Department\_Summary;

**Expected Output:**

| **department** | **employee\_count** |
| --- | --- |
| HR | 4 |
| IT | 3 |
| Finance | 2 |

**Exercise 16: Limit Login Attempts**

**Task:** Lock a user’s account after 3 consecutive failed login attempts.

**Step 1: Add a Status Column to Users Table**

ALTER TABLE Users ADD COLUMN account\_status VARCHAR(20) DEFAULT 'ACTIVE';

**Trigger to Lock Account:**

DELIMITER $$

CREATE TRIGGER lock\_account\_after\_failed\_attempts

AFTER INSERT ON Login\_Attempts

FOR EACH ROW

BEGIN

IF (SELECT COUNT(\*) FROM Login\_Attempts

WHERE user\_id = NEW.user\_id AND attempt\_status = 'FAILED'

AND attempt\_time >= NOW() - INTERVAL 1 DAY) >= 3 THEN

UPDATE Users SET account\_status = 'LOCKED' WHERE user\_id = NEW.user\_id;

END IF;

END$$

DELIMITER ;

**Test It:**

INSERT INTO Login\_Attempts (user\_id, attempt\_status) VALUES (1, 'FAILED');

INSERT INTO Login\_Attempts (user\_id, attempt\_status) VALUES (1, 'FAILED');

INSERT INTO Login\_Attempts (user\_id, attempt\_status) VALUES (1, 'FAILED');

SELECT \* FROM Users WHERE user\_id = 1;

**Expected Output:**

| **user\_id** | **username** | **account\_status** |
| --- | --- | --- |
| 1 | john\_doe | LOCKED |

**Exercise 17: Prevent Duplicate Salary for Employees in the Same Department**

**Task:** Ensure that no two employees in the same department have the same salary.

**Trigger to Enforce Unique Salaries per Department:**

DELIMITER $$

CREATE TRIGGER prevent\_duplicate\_salaries

BEFORE INSERT ON Employees

FOR EACH ROW

BEGIN

IF EXISTS (

SELECT 1 FROM Employees

WHERE department = NEW.department AND salary = NEW.salary

) THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Duplicate salary detected within the same department!';

END IF;

END$$

DELIMITER ;

**Test It:**

INSERT INTO Employees (emp\_name, department, salary) VALUES ('Laura', 'IT', 7500);

**Expected Output (if someone in IT already has 7500):**

ERROR 1644 (45000): Duplicate salary detected within the same department!

**Exercise 18: Track Unauthorized Deletion Attempts**

**Task:** Log unauthorized attempts to delete employees with critical roles (e.g., 'CEO' or 'CTO').

**Step 1: Create an Unauthorized Deletion Log Table**

CREATE TABLE Unauthorized\_Delete\_Log (

attempt\_id INT AUTO\_INCREMENT PRIMARY KEY,

emp\_id INT,

emp\_name VARCHAR(50),

attempted\_by VARCHAR(50),

attempt\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

**Trigger to Log Unauthorized Attempts:**

DELIMITER $$

CREATE TRIGGER log\_unauthorized\_deletion

BEFORE DELETE ON Employees

FOR EACH ROW

BEGIN

IF OLD.department IN ('Executive') THEN

INSERT INTO Unauthorized\_Delete\_Log (emp\_id, emp\_name, attempted\_by)

VALUES (OLD.emp\_id, OLD.emp\_name, CURRENT\_USER());

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Unauthorized attempt to delete an executive!';

END IF;

END$$

DELIMITER ;

**Test It:**

DELETE FROM Employees WHERE department = 'Executive';

**Expected Output:**

ERROR 1644 (45000): Unauthorized attempt to delete an executive!

Check the log:

SELECT \* FROM Unauthorized\_Delete\_Log;