-- Create table EMPLOYEES

CREATE TABLE EMPLOYEES (

EMP\_ID INT PRIMARY KEY,

EMP\_NAME VARCHAR(100),

DEPARTMENT VARCHAR(50),

SALARY INT

);

-- 1) Procedure to insert employee details

DELIMITER //

CREATE PROCEDURE add\_employee (

IN id INT,

IN name VARCHAR(100),

IN dept VARCHAR(50),

IN sal INT

)

BEGIN

INSERT INTO EMPLOYEES (EMP\_ID, EMP\_NAME, DEPARTMENT, SALARY)

VALUES (id, name, dept, sal);

END //

DELIMITER ;

-- Truncate the table before inserting data

TRUNCATE TABLE EMPLOYEES;

-- Insert sample employee data

CALL add\_employee(1, 'Sneha', 'IT', 50000);

CALL add\_employee(2, 'Abi', 'Sales', 30000);

CALL add\_employee(3, 'Ram', 'Management', 70000);

CALL add\_employee(4, 'Preetha', 'Testing', 40000);

-- Display all employees

SELECT \* FROM EMPLOYEES;

-- 2) Procedure to update salary based on conditions

DELIMITER //

CREATE PROCEDURE modify\_salary (

IN id INT

)

BEGIN

DECLARE sal INT;

SELECT SALARY INTO sal FROM EMPLOYEES WHERE EMP\_ID = id LIMIT 1;

IF sal < 5000 THEN

UPDATE EMPLOYEES SET SALARY = sal \* 1.10 WHERE EMP\_ID = id;

ELSEIF sal BETWEEN 5000 AND 10000 THEN

UPDATE EMPLOYEES SET SALARY = sal \* 1.075 WHERE EMP\_ID = id;

ELSE

UPDATE EMPLOYEES SET SALARY = sal \* 1.05 WHERE EMP\_ID = id;

END IF;

END //

DELIMITER ;

-- Enable SQL safe updates

SET SQL\_SAFE\_UPDATES = 1;

-- Update salary for employees

CALL modify\_salary(1);

CALL modify\_salary(2);

-- Display updated employees

SELECT \* FROM EMPLOYEES;

-- 3) Procedure to display employee names using a cursor

DELIMITER //

CREATE PROCEDURE show\_employee\_names()

BEGIN

DECLARE emp\_name VARCHAR(100);

DECLARE finished INT DEFAULT 0;

DECLARE emp\_cursor CURSOR FOR SELECT EMP\_NAME FROM EMPLOYEES;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET finished = 1;

CREATE TEMPORARY TABLE temp\_emp\_names (name VARCHAR(100));

OPEN emp\_cursor;

read\_loop: LOOP

FETCH emp\_cursor INTO emp\_name;

IF finished THEN

LEAVE read\_loop;

END IF;

INSERT INTO temp\_emp\_names (name) VALUES (emp\_name);

END LOOP;

CLOSE emp\_cursor;

SELECT \* FROM temp\_emp\_names;

DROP TEMPORARY TABLE temp\_emp\_names;

END //

DELIMITER ;

-- Call procedure to display employee names

CALL show\_employee\_names();

-- 4) Create view for employees with high salary

CREATE VIEW top\_earners AS

SELECT EMP\_ID, EMP\_NAME, DEPARTMENT, SALARY

FROM EMPLOYEES

WHERE SALARY > 30000;

-- Display view data

SELECT \* FROM top\_earners;

-- 5) Function to calculate bonus based on salary

DELIMITER //

CREATE FUNCTION calc\_bonus (id INT) RETURNS DECIMAL(10,2)

READS SQL DATA

BEGIN

DECLARE sal INT;

DECLARE bonus DECIMAL(10,2);

SELECT SALARY INTO sal FROM EMPLOYEES WHERE EMP\_ID = id LIMIT 1;

IF sal < 5000 THEN

SET bonus = sal \* 0.10;

ELSEIF sal BETWEEN 5000 AND 10000 THEN

SET bonus = sal \* 0.075;

ELSE

SET bonus = sal \* 0.05;

END IF;

RETURN bonus;

END //

DELIMITER ;

-- Calculate bonus for employee with ID 1

SELECT calc\_bonus(1) AS bonus\_emp\_1;

-- 6) Create employee log table and trigger

CREATE TABLE EMPLOYEE\_LOG (

LOG\_ID INT AUTO\_INCREMENT PRIMARY KEY,

EMP\_ID INT,

EMP\_NAME VARCHAR(100),

DEPARTMENT VARCHAR(50),

SALARY INT,

INSERTED\_AT TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

DELIMITER //

CREATE TRIGGER log\_employee\_addition

AFTER INSERT ON EMPLOYEES

FOR EACH ROW

BEGIN

INSERT INTO EMPLOYEE\_LOG (EMP\_ID, EMP\_NAME, DEPARTMENT, SALARY)

VALUES (NEW.EMP\_ID, NEW.EMP\_NAME, NEW.DEPARTMENT, NEW.SALARY);

END //

DELIMITER ;

-- Display employee log

SELECT \* FROM EMPLOYEE\_LOG;

-- 7) Setup for customer and order tables

DROP TABLE IF EXISTS customers, orders, orderitems;

CREATE TABLE customers (

customerid INT PRIMARY KEY,

customer\_name VARCHAR(100),

credit\_limit DECIMAL(10, 2) DEFAULT 0

);

CREATE TABLE orders (

orderid INT PRIMARY KEY,

customerid INT,

status VARCHAR(50),

salesmanid INT,

order\_date DATE,

FOREIGN KEY (customerid) REFERENCES customers(customerid)

);

CREATE TABLE orderitems (

orderid INT,

itemid INT,

productid INT,

quantity INT,

unit\_price DECIMAL(10, 2),

PRIMARY KEY (orderid, itemid),

FOREIGN KEY (orderid) REFERENCES orders(orderid)

);

-- Insert sample data into customers

INSERT INTO customers (customerid, customer\_name) VALUES

(1, 'Ravi'),

(2, 'Suresh'),

(3, 'Mani');

-- Insert sample data into orders

INSERT INTO orders (orderid, customerid, status, salesmanid, order\_date) VALUES

(1, 1, 'Shipped', 101, '2023-01-15'),

(2, 2, 'Shipped', 102, '2023-02-20'),

(3, 1, 'Pending', 103, '2023-03-05'),

(4, 3, 'Shipped', 101, '2023-04-10');

-- Insert sample data into orderitems

INSERT INTO orderitems (orderid, itemid, productid, quantity, unit\_price) VALUES

(1, 1, 1001, 2, 500.00),

(1, 2, 1002, 1, 1500.00),

(2, 1, 1001, 3, 500.00),

(3, 1, 1003, 4, 250.00),

(4, 1, 1002, 2, 1500.00);

-- Create view for sales revenue by customers

CREATE VIEW customer\_sales\_revenue AS

SELECT

o.customerid,

SUM(oi.quantity \* oi.unit\_price) AS total\_revenue,

SUM(oi.quantity \* oi.unit\_price) \* 0.05 AS credit

FROM

orders o

JOIN

orderitems oi ON o.orderid = oi.orderid

GROUP BY

o.customerid;

DELIMITER //

CREATE PROCEDURE adjust\_credit\_limits()

BEGIN

DECLARE budget DECIMAL(10, 2) DEFAULT 1000000;

DECLARE remaining\_budget DECIMAL(10, 2) DEFAULT 1000000;

DECLARE credit DECIMAL(10, 2);

DECLARE cust\_id INT;

DECLARE total\_sales DECIMAL(10, 2);

DECLARE done INT DEFAULT 0;

DECLARE cursor\_customers CURSOR FOR

SELECT customerid, total\_revenue

FROM customer\_sales\_revenue

ORDER BY total\_revenue DESC;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

UPDATE customers SET credit\_limit = 0;

OPEN cursor\_customers;

loop\_customers: LOOP

FETCH cursor\_customers INTO cust\_id, total\_sales;

IF done THEN

LEAVE loop\_customers;

END IF;

SET credit = total\_sales \* 0.05;

IF credit > remaining\_budget THEN

SET credit = remaining\_budget;

END IF;

UPDATE customers SET credit\_limit = credit WHERE customerid = cust\_id;

SET remaining\_budget = remaining\_budget - credit;

IF remaining\_budget <= 0 THEN

LEAVE loop\_customers;

END IF;

END LOOP;

CLOSE cursor\_customers;

END //

DELIMITER ;

-- Call procedure to adjust credit limits

CALL adjust\_credit\_limits();

-- 8) Create and populate employees table

CREATE TABLE employees\_detail (

emp\_id INT PRIMARY KEY,

first\_name VARCHAR(25),

last\_name VARCHAR(25),

email VARCHAR(50),

phone VARCHAR(15),

hire\_date DATE,

job\_code VARCHAR(25),

salary INT,

commission\_pct DECIMAL(5,2),

mgr\_id INT,

dept\_id INT

);

INSERT INTO employees\_detail (emp\_id, first\_name, last\_name, email, phone, hire\_date, job\_code, salary, commission\_pct, mgr\_id, dept\_id)

VALUES

(1, 'Sneha', 'Kumar', 'sneha.abc@gmail.com', '123-456-7890', '2020-01-15', 'IT\_PROG', 50000, NULL, 101, 10),

(2, 'Abi', 'Shankar', 'abi.abc@gmail.com', '987-654-3210', '2021-02-20', 'SALES\_REP', 30000, 0.10, 102, 20),

(3, 'Ram', 'Raju', 'ram.abc@gmail.com', '555-123-4567', '2022-03-05', 'MGR', 70000, NULL, NULL, 30),

(4, 'Preetha', 'Selvan', 'preetha.abc@gmail.com', '555-987-6543', '2023-04-10', 'TEST\_ENG', 40000, NULL, 103, 40);

-- Display employee details

SELECT \* FROM employees\_detail;

-- Create job table and insert data

CREATE TABLE jobs (

job\_code VARCHAR(25) PRIMARY KEY,

job\_title VARCHAR(25)

);

INSERT INTO jobs (job\_code, job\_title)

VALUES

('IT\_PROG', 'Programmer'),

('SALES\_REP', 'Sales Representative'),

('MGR', 'Manager'),

('TEST\_ENG', 'Test Engineer');

-- 9) Display job codes and job titles

SELECT \* FROM jobs;

-- 10) Create procedure to display employee details

DELIMITER //

CREATE PROCEDURE get\_employee\_detail()

BEGIN

SELECT

e.emp\_id,

e.first\_name,

e.last\_name,

e.email,

e.phone,

e.hire\_date,

j.job\_title,

e.salary,

e.commission\_pct,

e.mgr\_id,

e.dept\_id

FROM

employees\_detail e

JOIN

jobs j ON e.job\_code = j.job\_code;

END //

DELIMITER ;

-- Call procedure to display employee details

CALL get\_employee\_detail();