Downloaded from: justpaste.it/f330p dbms queries 1.Banking System Creating table with following attributes. Query: create table accd(accno int,customername varchar(60),balance int,levels varchar(60)) Inserting values into the table. Query: insert into accd values('012','karthi','10000',''),('345','vishal','20000',''), ('678','sanjeev','60000','') Creating procedure for deposit. delimiter $$ create procedure deposit( num int, bal int) BEGIN declare bal1 int; update accd set balance = balance+bal where accno=num; select balance into bal1 from accd where accno=num; if (bal1>=0 and bal1=10000 and bal1=50000 then update accd set levels ='platinum' where accno=num; else update accd set levels='null' where accno=num; end if; select \* from accd; end Creating procedure for withdrawl. delimiter $$ create procedure withdrawl( num int, bal int) BEGIN declare bal1 int; update accd set balance = balance-bal where accno=num; select balance into bal1 from accd where accno=num; if (bal1>=0 and bal1=10000 and bal1=50000 then update accd set levels ='platinum' where accno=num; else update accd set levels='null' where accno=num; end if; select \* from accd; END 2. Hospital Management CREATE TABLE patients (patient\_id INT PRIMARY KEY, patient\_name VARCHAR(60), age INT, gender VARCHAR(10), room\_no INT,status VARCHAR(20)); CREATE TABLE rooms (room\_no INT PRIMARY KEY, room\_type VARCHAR(20),availability VARCHAR(10)); INSERT INTO patients VALUES (1, 'John Doe', 30, 'Male', 101, 'Admitted'), (2, 'Jane Smith', 25, 'Female', 102, 'Discharged'), (3, 'Emily Davis', 40, 'Female', 103, 'Admitted'); INSERT INTO rooms VALUES (101, 'General', 'Occupied'), (102, 'Private', 'Available'), (103, 'ICU', 'Occupied'); DELIMITER $$ CREATE PROCEDURE admit\_patient( IN p\_id INT, IN p\_name VARCHAR(60), IN p\_age INT, IN p\_gender VARCHAR(10), IN r\_no INT) BEGIN DECLARE r\_availability VARCHAR(10); -- Check room availability SELECT availability INTO r\_availability FROM rooms WHERE room\_no = r\_no; IF r\_availability = 'Available' THEN -- Insert patient details into patients table INSERT INTO patients (patient\_id, patient\_name, age, gender, room\_no, status) VALUES (p\_id, p\_name, p\_age, p\_gender, r\_no, 'Admitted'); -- Update room availability UPDATE rooms SET availability = 'Occupied' WHERE room\_no = r\_no; ELSE SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Room is not available'; END IF; -- Display updated patient and room details SELECT \* FROM patients; SELECT \* FROM rooms; END $$ DELIMITER ; DELIMITER $$ CREATE PROCEDURE discharge\_patient( IN p\_id INT) BEGIN DECLARE r\_no INT; -- Get room number of the patient SELECT room\_no INTO r\_no FROM patients WHERE patient\_id = p\_id; -- Update patient status to Discharged UPDATE patients SET status = 'Discharged' WHERE patient\_id = p\_id; -- Update room availability UPDATE rooms SET availability = 'Available' WHERE room\_no = r\_no; -- Display updated patient and room details SELECT \* FROM patients; SELECT \* FROM rooms; END $$ DELIMITER ; 3.Airline Reservation CREATE TABLE flights ( flight\_id INT PRIMARY KEY, flight\_name VARCHAR(60), total\_seats INT, available\_seats INT ); CREATE TABLE passengers ( passenger\_id INT PRIMARY KEY, passenger\_name VARCHAR(60), flight\_id INT, status VARCHAR(20), FOREIGN KEY (flight\_id) REFERENCES flights(flight\_id) ); INSERT INTO flights VALUES (1, 'Flight A', 100, 100), (2, 'Flight B', 200, 200), (3, 'Flight C', 150, 150); INSERT INTO passengers VALUES (1, 'Alice', 1, 'Booked'), (2, 'Bob', 2, 'Cancelled'), (3, 'Charlie', 3, 'Booked'); DELIMITER $$ CREATE PROCEDURE book\_flight( IN p\_id INT, IN p\_name VARCHAR(60), IN f\_id INT) BEGIN DECLARE seats\_available INT; DECLARE passenger\_exists INT; -- Check if the passenger already exists SELECT COUNT(\*) INTO passenger\_exists FROM passengers WHERE passenger\_id = p\_id; IF passenger\_exists > 0 THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Passenger already exists'; ELSE -- Check seat availability SELECT available\_seats INTO seats\_available FROM flights WHERE flight\_id = f\_id; IF seats\_available > 0 THEN -- Insert passenger details into passengers table INSERT INTO passengers (passenger\_id, passenger\_name, flight\_id, status) VALUES (p\_id, p\_name, f\_id, 'Booked'); -- Update available seats UPDATE flights SET available\_seats = available\_seats - 1 WHERE flight\_id = f\_id; ELSE SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'No available seats'; END IF; END IF; -- Display updated flight and passenger details SELECT \* FROM passengers; SELECT \* FROM flights; END $$ DELIMITER ; DELIMITER $$ CREATE PROCEDURE cancel\_booking( IN p\_id INT) BEGIN DECLARE f\_id INT; DECLARE passenger\_exists INT; -- Check if the passenger exists SELECT COUNT(\*) INTO passenger\_exists FROM passengers WHERE passenger\_id = p\_id; IF passenger\_exists = 0 THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Passenger does not exist'; ELSE -- Get flight id of the passenger SELECT flight\_id INTO f\_id FROM passengers WHERE passenger\_id = p\_id; -- Update passenger status to Cancelled UPDATE passengers SET status = 'Cancelled' WHERE passenger\_id = p\_id; -- Update available seats UPDATE flights SET available\_seats = available\_seats + 1 WHERE flight\_id = f\_id; END IF; -- Display updated passenger and flight details SELECT \* FROM passengers; SELECT \* FROM flights; END $$ DELIMITER ; 4.Payroll CREATE TABLE employees ( emp\_id INT PRIMARY KEY, emp\_name VARCHAR(60), sala ry INT, level VARCHAR(60) ); INSERT INTO employees VALUES (1, 'John Doe', 50000, ''), (2, 'Jane Smith', 70000, ''), (3, 'Alice Johnson', 30000, ''); DELIMITER $$ CREATE PROCEDURE deposit\_salary( IN emp\_id INT, IN amount INT) BEGIN DECLARE new\_salary INT; -- Update salary UPDATE employees SET salary = salary + amount WHERE emp\_id = emp\_id; -- Get the updated salary SELECT salary INTO new\_salary FROM employees WHERE emp\_id = emp\_id; -- Update level based on the updated salary IF new\_salary >= 0 AND new\_salary < 30000 THEN UPDATE employees SET level = 'Junior' WHERE emp\_id = emp\_id; ELSEIF new\_salary >= 30000 AND new\_salary < 60000 THEN UPDATE employees SET level = 'Mid' WHERE emp\_id = emp\_id; ELSEIF new\_salary >= 60000 THEN UPDATE employees SET level = 'Senior' WHERE emp\_id = emp\_id; ELSE UPDATE employees SET level = 'Unknown' WHERE emp\_id = emp\_id; END IF; -- Display updated employee details SELECT \* FROM employees; END $$ DELIMITER ; DELIMITER $$ CREATE PROCEDURE deduct\_salary( IN emp\_id INT, IN amount INT) BEGIN DECLARE new\_salary INT; -- Update salary UPDATE employees SET salary = salary - amount WHERE emp\_id = emp\_id; -- Get the updated salary SELECT salary INTO new\_salary FROM employees WHERE emp\_id = emp\_id; -- Update level based on the updated salary IF new\_salary >= 0 AND new\_salary < 30000 THEN UPDATE employees SET level = 'Junior' WHERE emp\_id = emp\_id; ELSEIF new\_salary >= 30000 AND new\_salary < 60000 THEN UPDATE employees SET level = 'Mid' WHERE emp\_id = emp\_id; ELSEIF new\_salary >= 60000 THEN UPDATE employees SET level = 'Senior' WHERE emp\_id = emp\_id; ELSE UPDATE employees SET level = 'Unknown' WHERE emp\_id = emp\_id; END IF; -- Display updated employee details SELECT \* FROM employees; END $$ DELIMITER ; 5.Subject Allocation CREATE TABLE students ( student\_id INT PRIMARY KEY, student\_name VARCHAR(60) ); CREATE TABLE subjects ( subject\_id INT PRIMARY KEY, subject\_name VARCHAR(60) ); CREATE TABLE subject\_allocation ( allocation\_id INT AUTO\_INCREMENT PRIMARY KEY, st udent\_id INT, subject\_id INT, FOREIGN KEY (student\_id) REFERENCES students(student\_id ), FOREIGN KEY (subject\_id) REFERENCES subjects(subject\_id) ); INSERT INTO students VALUES (1, 'John Doe'), (2, 'Jane Smith'), (3, 'Alice Johnson'); INSERT INTO subjects VALUES (101, 'Mathematics'), (102, 'Physics'), (103, 'Chemistry'); DELIMITER $$ CREATE PROCEDURE assign\_subject( IN student\_id INT, IN subject\_id INT) BEGIN DECLARE subject\_exists INT; DECLARE allocation\_exists INT; -- Check if the subject exists for the student SELECT COUNT(\*) INTO subject\_exists FROM subjects WHERE subject\_id = subject\_id; SELECT COUNT(\*) INTO allocation\_exists FROM subject\_allocation WHERE student\_id = student\_id AND subject\_id = subject\_id; IF subject\_exists = 0 THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Subject does not exist'; ELSEIF allocation\_exists > 0 THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Subject already assigned to student'; ELSE -- Insert allocation details into subject\_allocation table INSERT INTO subject\_allocation (student\_id, subject\_id) VALUES (student\_id, subject\_id); END IF; -- Display updated subject allocation details SELECT \* FROM subject\_allocation; END $$ DELIMITER ; DELIMITER $$ CREATE PROCEDURE remove\_subject( IN student\_id INT, IN subject\_id INT) BEGIN DECLARE allocation\_exists INT; -- Check if the allocation exists SELECT COUNT(\*) INTO allocation\_exists FROM subject\_allocation WHERE student\_id = student\_id AND subject\_id = subject\_id; IF allocation\_exists = 0 THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Subject not assigned to student'; ELSE -- Delete the allocation details from subject\_allocation table DELETE FROM subject\_allocation WHERE student\_id = student\_id AND subject\_id = subject\_id; END IF; -- Display updated subject allocation details SELECT \* FROM subject\_allocation; END $$ DELIMITER ; 6.Hr recruitment CREATE TABLE workers ( emp\_id INT PRIMARY KEY, emp\_name VARCHAR(60), salary INT, status VARCHAR(60), level VARCHAR(60) ); INSERT INTO workers VALUES (1, 'John Doe', 30000, 'Active', ''), (2, 'Jane Smith', 50000, 'Active', ''), (3, 'Alice Johnson', 70000, 'Active', ''); DELIMITER $$ CREATE PROCEDURE hire\_worker( IN worker\_id INT, IN worker\_name VARCHAR(60), IN salary INT) BEGIN DECLARE worker\_exists INT; -- Check if the worker already exists SELECT COUNT(\*) INTO worker\_exists FROM workers WHERE emp\_id = worker\_id; IF worker\_exists > 0 THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Worker already exists'; ELSE -- Insert worker details into workers table INSERT INTO workers (emp\_id, emp\_name, salary, status, level) VALUES (worker\_id, worker\_name, salary, 'Active', ''); -- Update level based on salary IF salary >= 0 AND salary < 30000 THEN UPDATE workers SET level = 'Junior' WHERE emp\_id = worker\_id; ELSEIF salary >= 30000 AND salary < 60000 THEN UPDATE workers SET level = 'Mid' WHERE emp\_id = worker\_id; ELSEIF salary >= 60000 THEN UPDATE workers SET level = 'Senior' WHERE emp\_id = worker\_id; ELSE UPDATE workers SET level = 'Unknown' WHERE emp\_id = worker\_id; END IF; END IF; -- Display updated worker details SELECT \* FROM workers; END $$ DELIMITER ; DELIMITER $$ CREATE PROCEDURE terminate\_worker( IN worker\_id INT) BEGIN DECLARE worker\_exists INT; -- Check if the worker exists SELECT COUNT(\*) INTO worker\_exists FROM workers WHERE emp\_id = worker\_id; IF worker\_exists = 0 THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Worker does not exist'; ELSE -- Update worker status to Terminated UPDATE workers SET status = 'Terminated', level = 'N/A' WHERE emp\_id = worker\_id; END IF; -- Display updated worker details SELECT \* FROM workers; END $$ DELIMITER ; 7.Department Store Maintainence CREATE TABLE inventory ( item\_id INT PRIMARY KEY, item\_name VARCHAR(60), stock INT , status VARCHAR(60) ); INSERT INTO inventory VALUES (1, 'Shampoo', 50, ''), (2, 'Soap', 30, ''), (3, 'Toothpaste', 100, ''); DELIMITER $$ CREATE PROCEDURE add\_inventory( IN item\_id INT, IN quantity INT) BEGIN DECLARE current\_stock INT; -- Update inventory stock UPDATE inventory SET stock = stock + quantity WHERE item\_id = item\_id; -- Get the updated stock level SELECT stock INTO current\_stock FROM inventory WHERE item\_id = item\_id; -- Update status based on stock level IF current\_stock < 10 THEN UPDATE inventory SET status = 'Low Stock' WHERE item\_id = item\_id; ELSEIF current\_stock >= 10 AND current\_stock < 50 THEN UPDATE inventory SET status = 'In Stock' WHERE item\_id = item\_id; ELSEIF current\_stock >= 50 THEN UPDATE inventory SET status = 'Overstock' WHERE item\_id = item\_id; ELSE UPDATE inventory SET status = 'Unknown' WHERE item\_id = item\_id; END IF; -- Display updated inventory details SELECT \* FROM inventory; END $$ DELIMITER ; DELIMITER $$ CREATE PROCEDURE sell\_item( IN item\_id INT, IN quantity INT) BEGIN DECLARE current\_stock INT; -- Update inventory stock UPDATE inventory SET stock = stock - quantity WHERE item\_id = item\_id; -- Get the updated stock level SELECT stock INTO current\_stock FROM inventory WHERE item\_id = item\_id; -- Update status based on stock level IF current\_stock < 10 THEN UPDATE inventory SET status = 'Low Stock' WHERE item\_id = item\_id; ELSEIF current\_stock >= 10 AND current\_stock < 50 THEN UPDATE inventory SET status = 'In Stock' WHERE item\_id = item\_id; ELSEIF current\_stock >= 50 THEN UPDATE inventory SET status = 'Overstock' WHERE item\_id = item\_id; ELSE UPDATE inventory SET status = 'Unknown' WHERE item\_id = item\_id; END IF; -- Display updated inventory details SELECT \* FROM inventory; END $$ DELIMITER ; 8.Sports event conduction CREATE TABLE participants ( participant\_id INT PRIMARY KEY, participant\_name VARCHAR (60), score INT, level VARCHAR(60) ); INSERT INTO participants VALUES (1, 'Alice', 0, ''), (2, 'Bob', 0, ''), (3, 'Charlie', 0, ''); DELIMITER $$ CREATE PROCEDURE register\_participant( IN part\_id INT, IN part\_name VARCHAR(60)) BEGIN DECLARE participant\_exists INT; -- Check if the participant already exists SELECT COUNT(\*) INTO participant\_exists FROM participants WHERE participant\_id = part\_id; IF participant\_exists > 0 THEN SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Participant already exists'; ELSE -- Insert participant details into participants table INSERT INTO participants (participant\_id, participant\_name, score, level) VALUES (part\_id, part\_name, 0, 'Beginner'); END IF; -- Display updated participants details SELECT \* FROM participants; END $$ DELIMITER ; DELIMITER $$ CREATE PROCEDURE update\_score( IN part\_id INT, IN new\_score INT) BEGIN DECLARE current\_score INT; -- Update participant's score UPDATE participants SET score = new\_score WHERE participant\_id = part\_id; -- Get the updated score SELECT score INTO current\_score FROM participants WHERE participant\_id = part\_id; -- Update level based on score IF current\_score >= 0 AND current\_score < 50 THEN UPDATE participants SET level = 'Beginner' WHERE participant\_id = part\_id; ELSEIF current\_score >= 50 AND current\_score < 100 THEN UPDATE participants SET level = 'Intermediate' WHERE participant\_id = part\_id; ELSEIF current\_score >= 100 THEN UPDATE participants SET level = 'Advanced' WHERE participant\_id = part\_id; ELSE UPDATE participants SET level = 'Unknown' WHERE participant\_id = part\_id; END IF; -- Display updated participants details SELECT \* FROM participants; END $$ DELIMITER ;