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) then update accd set levels ='silver' where accno=num;
elseif (bal1>=10000 and bal1<50000) then update accd set levels ='gold' where
accno=num;
elseif 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) then update accd set levels ='silver' where accno=num;
elseif (bal1>=10000 and bal1<50000) then update accd set levels ='gold' where
accno=num;
elseif 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 ;
```