Stored Procedures

```
create table emp (
empID integer,
fName varchar(20),
lName varchar(20),
salary float,
primary key(empID)
)

insert into emp values(100,'Mali','Kevin',100000);
insert into emp values(101,'Vivi','De Silva',120000);
insert into emp values(103,'Nadun','Ranmuthu',110000);
insert into emp values(107,'Dev','Steven',140000);
insert into emp values(110,'Kithmi','Perera',150000);
```

1.Create a Stored Procedure to display all employee details.

```
CREATE PROCEDURE ShowAllEmployees()
BEGIN
  SELECT * FROM emp;
END
```

2. Create a Stored Procedure to display all employee details whose salary is greater than 100 000.

```
CREATE PROCEDURE ShowHighSalaryEmployees()
BEGIN
  SELECT * FROM emp WHERE salary > 100000;
END
```

3 Create a Stored Procedure to display all employee details whose salary is greater than 100 000.

```
CREATE PROCEDURE GetEmployeeByID(IN emp_id INT)
BEGIN
  SELECT * FROM emp WHERE empID = emp_id;
END
```

4. Create a Stored Procedure to display employee details for the given salary. (Hint: use input parameters)

```
CREATE PROCEDURE GetEmployeeBySalary(IN sal FLOAT)
BEGIN
  SELECT * FROM emp WHERE salary = sal;
END
```

5.Create a Stored Procedure to display salary of a given employee. (Hint: use input and output parameters)

```
CREATE PROCEDURE GetSalaryByEmpID(
  IN emp_id INT,
  OUT emp_salary FLOAT
)
BEGIN
  SELECT salary INTO emp_salary FROM emp WHERE empID = emp_id;
END
```

6.Create a stored procedure to insert a new employee record to the above created table. Procedure should check whether the employee name exists in the table. If it exists it should print an error message, otherwise, record should be inserted.

```
CREATE PROCEDURE AddEmployeeIfNotExists(
  IN newID INT,
  IN fname VARCHAR(20),
  IN lname VARCHAR(20),
  IN sal FLOAT
)
BEGIN
  DECLARE count_name INT;

  SELECT COUNT(*) INTO count_name
  FROM emp
  WHERE fName = fname AND lName = lname;

  IF count_name > 0 THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Employee name already exists.';
  ELSE
    INSERT INTO emp VALUES(newID, fname, lname, sal);
  END IF;
END
```

7.Execute the stored procedure you created in activity 2 to display all employee details.

```
CALL ShowAllEmployees();
```

8. Create a stored procedure to insert a new employee record to the above created table with following conditions.
The new employee ID should be one greater than the largest employee ID in the table. (e.g. the next ID is 104 which is 103+1). If the table does not have any data then the first employee ID should start at 100

```
CREATE PROCEDURE AddEmployeeAutoID(
  IN fname VARCHAR(20),
  IN lname VARCHAR(20),
  IN sal FLOAT
)
BEGIN
  DECLARE max_id INT;
  DECLARE new_id INT;

  SELECT MAX(empID) INTO max_id FROM emp;

  IF max_id IS NULL THEN
    SET new_id = 100;
  ELSE
    SET new_id = max_id + 1;
  END IF;

  INSERT INTO emp VALUES(new_id, fname, lname, sal);
END
```

9. Create a stored procedure to delete a given employee from the emp table.

```
CREATE PROCEDURE DeleteEmployeeByID(IN emp_id INT)
BEGIN
  DELETE FROM emp WHERE empID = emp_id;
END
```

10.Write a T-SQL function to return the number of employees have the salary above the given salary.

```
CREATE FUNCTION dbo.fn_EmployeesAboveSalary (@Salary FLOAT)
RETURNS INT
AS
BEGIN
  DECLARE @Count INT;

  SELECT @Count = COUNT(*)
  FROM emp
  WHERE salary > @Salary;

  RETURN @Count;
END;
```

11.Write the command(s) you would use to print the number of employees have the salary more than to 130000.Use the function you have created in the previous question

```
SELECT dbo.fn_EmployeesAboveSalary(130000) AS EmployeesAbove130k;
```

12 .Create a Stored Procedure to display all employee details.

```
SELECT dbo.fn_AverageSalary() AS AverageSalary;
```

Triggers

12. Create a Stored Procedure to display all employee details whose salary is greater than 100 000.

```
CREATE TABLE EmpLog (
  EmpId     INT,
  Log_date  DATE,
  New_salary FLOAT,
  Action    VARCHAR(20)
);
```

13. Create the following trigger 'log_salary_increase'. This trigger is fired when salary is updated. After the update occurs, this trigger will insert an entry to EmpLog table. Now update the emp table and view the EmpLog table.

Create the Trigger

```
CREATE TRIGGER log_salary_increase
ON emp
AFTER UPDATE
AS
BEGIN
  -- Insert into EmpLog only if salary is updated
  INSERT INTO EmpLog (EmpId, Log_date, New_salary, Action)
  SELECT
    i.empID,
    GETDATE(),
    i.salary,
    'Updated'
  FROM
    inserted i
  INNER JOIN deleted d ON i.empID = d.empID
  WHERE
    i.salary <> d.salary;
END;
```

Update the Employee Salary

```
UPDATE emp
SET salary = salary + 5000
WHERE empID = 100;
```

View the EmpLog Table

```
SELECT * FROM EmpLog;
```

14. Create the following trigger 'log_salary_increase'. This trigger is fired when salary is updated. After the update occurs, this trigger will insert an entry to EmpLog table. Now update the emp table and view the EmpLog table.

Create the Trigger

```
CREATE TRIGGER log_new_emp
ON emp
AFTER INSERT
AS
BEGIN
  INSERT INTO EmpLog (EmpId, Log_date, New_salary, Action)
  SELECT
    empID,
    GETDATE(),
    salary,
    'Inserted'
  FROM inserted;
END;
```

Insert New Employee Record

```
INSERT INTO emp VALUES (120, 'Kamal', 'Perera', 100000);
```

View the EmpLog Table

```
SELECT * FROM EmpLog;
```

14. Create the following trigger 'log_del_emp'. This trigger is fired when an employee is deleted. After the delete occurs, this trigger will insert an entry to EmpLog table.
Now delete from emp table and view the EmpLog table

Create the Trigger

```
CREATE TRIGGER log_del_emp
ON emp
AFTER DELETE
AS
BEGIN
  INSERT INTO EmpLog (EmpId, Log_date, New_salary, Action)
  SELECT
    empID,
    GETDATE(),
    salary,
    'Deleted'
  FROM deleted;
END;
```

Delete Employee Record

```
DELETE FROM emp WHERE empID = 100;
```

View the EmpLog Table

```
SELECT * FROM EmpLog;
```

Functions, Stored Procedures, View & Triggers

Consider the following schema.

Physician (eid, ename, position)
Department (did, name, head) foreign key (head) references Physician (eid)
Works_in (physician, department)

foreign key (physician) references Physician (eid)
foreign key (department) references Department (did)

Patient (pid, name, address, phone, insuranceId)
Appointments (appointmentId, patient, physician, startTime, endTime, room)

foreign key (physician) references Physician (eid)
foreign key (patient) references Patient (pid)

Drugs (Code, name, brand)
Prescribes (physician, patient, drug, date, appointment, dose)

foreign key (physician) references Physician (eid)
foreign key (patient) references Patient (pid)
foreign key (drug) references Drugs (Code)

Activity 1

1.Find the names of the physicians who does not head any department.

```
SELECT ename
FROM Physician
WHERE ename LIKE 'John %';
```

2.Find the names of the physicians who does not head any department.

```
SELECT ename
FROM Physician
WHERE eid NOT IN (
  SELECT head
  FROM Department
  WHERE head IS NOT NULL
);
```

3.Find the drug which is prescribed the most.

```
SELECT ename
FROM Physician
WHERE eid NOT IN (
  SELECT head
  FROM Department
  WHERE head IS NOT NULL
);
```

Activity 2 : Working with Functions

4.Write a T-SQL function to return the number of physicians in a given department.

```
CREATE FUNCTION dbo.fn_PhysiciansInDepartment (@deptId INT)
RETURNS INT
AS
BEGIN
  DECLARE @count INT

  SELECT @count = COUNT(*)
  FROM Works_in
  WHERE department = @deptId

  RETURN @count
END
```

5.Write the command(s) you would use to print the number of physicians in the 'Psychiatry' using the function you have created in the previous question

```
DECLARE @deptId INT

-- Get department ID for Psychiatry
SELECT @deptId = did
FROM Department
WHERE name = 'Psychiatry'

-- Use the function to get the count
SELECT dbo.fn_PhysiciansInDepartment(@deptId) AS NumberOfPhysicians
```

6.Write a T-SQL function that returns the number of appointments a given physicians have on a given date.

```
CREATE FUNCTION dbo.fn_AppointmentsOnDate (
  @physicianId INT,
  @appointmentDate DATE
)
RETURNS INT
AS
BEGIN
  DECLARE @count INT

  SELECT @count = COUNT(*)
  FROM Appointments
  WHERE physician = @physicianId
    AND CAST(startTime AS DATE) = @appointmentDate

  RETURN @count
END
```

Activity 3: Working with Stored-Procedures

7.Write a T-SQL procedure that outputs number of drugs prescribed to a given patient by a given doctor.

```
CREATE PROCEDURE dbo.sp_DrugsPrescribedByDoctorToPatient
  @physicianId INT,
  @patientId INT,
  @drugCount INT OUTPUT
AS
BEGIN
  SELECT @drugCount = COUNT(*)
  FROM Prescribes
  WHERE physician = @physicianId AND patient = @patientId
END
```

8.Write the command(s) you would use to print the number of drugs prescribed by the physician named 'Molly Clock' to patient named 'Dennis Doe' using the procedure you have created in the previous question.

```
DECLARE @physicianId INT, @patientId INT, @count INT

-- Get the physician ID for Molly Clock
SELECT @physicianId = eid
FROM Physician
WHERE ename = 'Molly Clock'

-- Get the patient ID for Dennis Doe
SELECT @patientId = pid
FROM Patient
WHERE name = 'Dennis Doe'

-- Call the stored procedure
EXEC dbo.sp_DrugsPrescribedByDoctorToPatient
  @physicianId = @physicianId,
  @patientId = @patientId,
  @drugCount = @count OUTPUT

-- Display the result
PRINT 'Number of drugs prescribed: ' + CAST(@count AS VARCHAR)
```

Activity 4: Working with Views

9.Create a view which shows a physician name, position, department name and the number of appointments for each physician.

```
CREATE VIEW vw_PhysicianAppointmentSummary AS
SELECT
  p.ename AS PhysicianName,
  p.position,
  d.name AS DepartmentName,
  COUNT(a.appointmentId) AS AppointmentCount
FROM Physician p
JOIN Works_in w ON p.eid = w.physician
JOIN Department d ON w.department = d.did
LEFT JOIN Appointments a ON p.eid = a.physician
GROUP BY p.ename, p.position, d.name;
```

Activity 5: Working with Triggers

10. Create a trigger that ensures that a doctor cannot have more than 3 appointments per a day. Add more sample data and check whether the trigger works as expected.

```
CREATE TRIGGER trg_LimitAppointmentsPerDay
ON Appointments
INSTEAD OF INSERT
AS
BEGIN
  DECLARE @physicianId INT, @apptDate DATE, @existingCount INT

  SELECT @physicianId = physician,
      @apptDate = CAST(startTime AS DATE)
  FROM INSERTED

  -- Count existing appointments for that physician on that date
  SELECT @existingCount = COUNT(*)
  FROM Appointments
  WHERE physician = @physicianId AND CAST(startTime AS DATE) = @apptDate

  -- Allow insert only if total (existing + new) ≤ 3
  IF @existingCount >= 3
  BEGIN
    RAISERROR('Doctor already has 3 appointments on this date.', 16, 1)
  END
  ELSE
  BEGIN
      INSERT INTO Appointments (appointmentId, patient, physician, startTime,
endTime, room)
    SELECT appointmentId, patient, physician, startTime, endTime, room
    FROM INSERTED
  END
END
```

Insert some appointments:

```
-- Assuming physician with ID = 1
INSERT INTO Appointments VALUES (201, 1, 1, '2025-06-22 09:00', '2025-06-22
09:30', 'R1');
INSERT INTO Appointments VALUES (202, 2, 1, '2025-06-22 10:00', '2025-06-22
10:30', 'R2');
INSERT INTO Appointments VALUES (203, 3, 1, '2025-06-22 11:00', '2025-06-22
11:30', 'R3');

-- This should fail (4th appointment on same day)
INSERT INTO Appointments VALUES (204, 4, 1, '2025-06-22 12:00', '2025-06-22
12:30', 'R4');
```

11.Create a trigger to ensure that the same drug is not prescribed to the same patient more than two times. Add more sample data and check whether the trigger works as intended.

```
CREATE TRIGGER trg_LimitDrugPrescriptions
ON Prescribes
INSTEAD OF INSERT
AS
BEGIN
      DECLARE @physicianId INT, @patientId INT, @drugCode
VARCHAR(20), @prescriptionCount INT

  SELECT @physicianId = physician,
      @patientId = patient,
      @drugCode = drug
  FROM INSERTED

  SELECT @prescriptionCount = COUNT(*)
  FROM Prescribes
  WHERE patient = @patientId AND drug = @drugCode

  IF @prescriptionCount >= 2
  BEGIN
      RAISERROR('This drug has already been prescribed to this
patient more than twice.', 16, 1)
  END
  ELSE
  BEGIN
        INSERT INTO Prescribes (physician, patient, drug, date,
appointment, dose)
    SELECT physician, patient, drug, date, appointment, dose
    FROM INSERTED
  END
END
```

Insert some appointments:

```
-- Assuming patient = 1 and drug = 'D01'
INSERT INTO Prescribes VALUES (1, 1, 'D01', '2025-06-22', 201, '10mg');
INSERT INTO Prescribes VALUES (1, 1, 'D01', '2025-06-23', 202, '10mg');

-- This should fail (3rd time)
INSERT INTO Prescribes VALUES (1, 1, 'D01', '2025-06-24', 203, '10mg');
```