## NIRAJ NANDISH

## 191IT234 | IT252 | 19/03/2021

# IT252

# Database Management System

## Assignment VII

1. Create stored procedure usp\_get\_employees\_salary\_above that accept a number as parameter and return all employees’ first and last names whose salary is above or equal to the given number. The result should be sorted by first\_name then by last\_name alphabetically.

> CREATE TABLE Employee(Emp\_number INT NOT NULL PRIMARY KEY, first\_name VARCHAR(30), last\_name VARCHAR(30), Salary INT);

>

> INSERT INTO Employee VALUES

> (1, 'William', 'Hartnell', 10000),

> (2, 'Patrick', 'Throughton', 20000),

> (3, 'Jon', 'Pertwee', 30000),

> (4, 'Tom', 'Baker', 40000),

> (5, 'Peter', 'Davison', 50000),

> (6, 'Colin', 'Baker', 60000),

> (7, 'Sylvester', 'McCoy', 70000),

> (8, 'Paul', 'McGann', 80000),

> (9, 'Christopher', 'Eccleston', 90000),

> (10, 'David', 'Tennant', 100000),

> (11, 'Matt', 'Smith', 110000),

> (12, 'Peter', 'Capaldi', 120000),

> (13, 'Jodie', 'Whittaker', 130000);

>

> DELIMITER $$

> CREATE PROCEDURE usp\_get\_employees\_salary\_above(IN sal INT)

> BEGIN

> SELECT first\_name, last\_name

> FROM Employee

> WHERE Salary >= sal

> ORDER BY first\_name, last\_name;

> END $$

> DELIMITER ;

Text

Description automatically generated> CALL usp\_get\_employees\_salary\_above(90000);

1. Write a stored procedure usp\_get\_towns\_starting\_with that accept string as parameter and returns all town names starting with that string. The result should be sorted by town name alphabetically.

> CREATE TABLE pgm2(Name VARCHAR(20));

>

> INSERT INTO pgm2 VALUES

> ('Bangalore'),

> ('Delhi'),

> ('Mysore'),

> ('Mangalore');

>

> DELIMITER $$

> CREATE PROCEDURE usp\_get\_towns\_starting\_with(prefix VARCHAR(20))

> BEGIN

> SELECT Name AS "town"

> FROM pgm2

> WHERE Name LIKE concat(prefix, "%")

> ORDER BY Name;

> END $$

> DELIMITER ;

> CALL usp\_get\_towns\_starting\_with("M");

Text

Description automatically generated

1. Write a function ufn\_get\_salary\_level that receives salary of an employee and returns the level of the salary. If salary is < 30000 return “Low”, if salary is between 30000 and 50000 (inclusive) return “Average” or If salary is > 50000 return “High”.

> DELIMITER $$

> CREATE FUNCTION ufn\_get\_salary\_level(sal INT)

> RETURNS VARCHAR(10)

> DETERMINISTIC

> BEGIN

> DECLARE lvl VARCHAR(10);

> IF sal < 30000 THEN

> SET lvl = 'Low';

> ELSEIF (sal >= 30000 AND sal <= 50000) THEN

> SET lvl = 'Average';

> ELSEIF sal > 50000 THEN

> SET lvl = 'High';

> END IF;

> RETURN (lvl);

> END $$

> DELIMITER ;

> SELECT ufn\_get\_salary\_level(20000);

> SELECT ufn\_get\_salary\_level(40000);

> SELECT ufn\_get\_salary\_level(60000);

Text

Description automatically generated

1. Create a function ufn\_calculate\_future\_value that accepts as parameters – sum, yearly interest rate and number of years. It should calculate and return the future value of the initial sum. Using the following formula: FV= I×((1+R)T), I – Initial sum, R – Yearly interest rate, T – Number of years .

> DELIMITER $$

> CREATE FUNCTION ufn\_calculate\_future\_value(p DECIMAL(8,2), r DECIMAL(5,2), t DECIMAL(5,2))

> RETURNS DECIMAL(8,2)

> DETERMINISTIC

> BEGIN

> DECLARE interest DECIMAL(8,2);

> SELECT POW(1+r, t) INTO interest;

> SET interest = p\*interest;

> RETURN (interest);

> END $$

> SELECT ufn\_calculate\_future\_value(1000, 0.1, 5);

Text

Description automatically generated

1. Create stored procedure Student\_Marks that accept a Roll\_number as parameter and return all students Name with their total marks whose total is above or equal to the given Roll number student Marks. The result should be reversely sorted by Total.

> CREATE TABLE Student(Roll\_number INTEGER NOT NULL PRIMARY KEY, Name VARCHAR(50), Total\_Marks INTEGER);

>

> INSERT INTO Student Values

> (34, 'Prasad', 460),

> (25, 'Gaurang', 465),

> (3, 'Binod', 450),

> (4, 'Pranav', 485),

> (30, 'Vijay', 490),

> (12, 'Kumar', 453),

> (8, 'Sanjay', 480),

> (6, 'Mihir', 440),

> (5, 'Nitya', 495),

> (15, 'Pushpa', 495),

> (21, 'Radhika', 455),

> (18, 'Mahika', 470),

> (11, 'Bhavika', 465);

>

> DELIMITER $$

> CREATE PROCEDURE Student\_Marks(IN rn INT)

> BEGIN

> SELECT Name, Total\_Marks

> FROM Student

> WHERE Total\_Marks >= (SELECT Total\_Marks FROM Student WHERE Roll\_number = rn)

> ORDER BY Total\_Marks DESC;

> END $$

> DELIMITER ;

Text

Description automatically generated> CALL Student\_Marks(8);

1. Write a function Get\_Grade\_level that receives Roll\_number of a student and returns the level of the marks of him / her. If Total\_Marks is < 300 return “C”, If Total\_Marks is between 300 and 400 (inclusive) return “B”, if Total\_Marks is between 401 and 450 (inclusive) > return “A”, if Total\_Marks is between 451 and 475 (inclusive)> return “A+” or if Total\_Marks is > 475 return “S”.

> DELIMITER $$

> CREATE FUNCTION Get\_Grade\_level(rn INT)

> RETURNS CHAR(2)

> DETERMINISTIC

> BEGIN

> DECLARE t\_marks INT;

> DECLARE lvl CHAR(2);

> SELECT Total\_Marks INTO t\_marks FROM Student WHERE Roll\_number = rn;

> IF t\_marks < 300 THEN

> SET lvl = "C";

> ELSEIF (t\_marks >= 300 AND t\_marks <= 400) THEN

> SET lvl = "B";

> ELSEIF (t\_marks > 400 AND t\_marks <= 450) THEN

> SET lvl = "A";

> ELSEIF (t\_marks > 450 AND t\_marks <= 475) THEN

> SET lvl = "A+";

> ELSEIF t\_marks > 475 THEN

> SET lvl = "S";

> END IF;

> RETURN (lvl);

> END $$

> DELIMITER ;

> SELECT Get\_Grade\_level(8);

Text

Description automatically generated

1. Write a function Get\_Sum that receives Limit as a parameter and returns sum of Natural numbers up to Limit.

> DELIMITER $$

> CREATE FUNCTION Get\_Sum(num INT)

> RETURNS INT

> DETERMINISTIC

> BEGIN

> RETURN (num\*(num+1)/2);

> END $$

> DELIMITER ;

> SELECT Get\_Sum(5);

Text

Description automatically generated

1. Write a function IT\_Rate\_employee that accepts a salary as input and returns Income-tax amount to the calling function as output. IT\_Rate Calculated based on the Salary is given below. If Basicpay is > 100000, IT = 20%, if Basicpay is between 100000 and 50000 (inclusive), IT = 15% or if Basicpay is < 50000, IT = 10% .

> DELIMITER $$

> CREATE FUNCTION IT\_Rate\_employee(sal INT)

> RETURNS DECIMAL(8,2)

> DETERMINISTIC

> BEGIN

> DECLARE it\_rate INT;

> IF sal > 100000 THEN

> SET it\_rate = 20;

> ELSEIF (sal <= 100000 AND sal >= 50000) THEN

> SET it\_rate = 15;

> ELSEIF sal < 50000 THEN

> SET it\_rate = 10;

> END IF;

> RETURN (sal \* it\_rate/100);

> END $$

> DELIMITER ;

Text

Description automatically generated> SELECT IT\_Rate\_employee(60000);

1. Create a Stored Procedure Net\_Pay\_employee that accepts a Emp\_no. as input parameter and returns Gross\_pay and Net\_Pay of all employees. The result should be sorted based on the Net\_Pay value (HRA= 12% of BP and DA=10% of BP).

**ASSUMPTION**: I used the same table as 1st Q, only modified it to the required schema.

> ALTER TABLE Employee RENAME COLUMN Salary TO Basicpay;

> ALTER TABLE Employee ADD Gross\_pay DECIMAL(8,2);

> ALTER TABLE Employee ADD Net\_pay DECIMAL(8,2);

>

> DELIMITER $$

> CREATE PROCEDURE Net\_Pay\_employee(IN emp\_no INT)

> BEGIN

> DECLARE sal INT;

> DECLARE it INT;

> DECLARE gp INT;

> DECLARE np INT;

> SELECT Basicpay INTO sal FROM Employee WHERE Emp\_number = emp\_no;

> SET it = IT\_Rate\_employee(sal);

> SET gp = sal + (sal\*0.12) + (sal\*0.1);

> SET np = gp - it;

> UPDATE Employee SET Gross\_pay = gp WHERE Emp\_number = emp\_no;

> UPDATE Employee SET Net\_pay = np WHERE Emp\_number = emp\_no;

> SELECT Gross\_pay, Net\_pay

> FROM Employee

> WHERE Gross\_pay IS NOT NULL AND Net\_pay IS NOT NULL

> ORDER BY Net\_pay DESC;

> END $$

> DELIMITER ;

> CALL Net\_Pay\_employee(1);

> CALL Net\_Pay\_employee(8);

> CALL Net\_Pay\_employee(12);

Text

Description automatically generated

1. Create a Stored Procedure for to implement anyone error/exception handler in Mysql.

> CREATE TABLE Books(BookID INT PRIMARY KEY, Title VARCHAR(50));

>

> DELIMITER $$

> CREATE PROCEDURE InsertBooks(IN inBookID INT, IN inTitle VARCHAR(50))

> BEGIN

> DECLARE EXIT HANDLER FOR 1062

> BEGIN

> SELECT CONCAT('Duplicate key: BookID ', inBookID, ' exists') AS message;

> END;

> DECLARE EXIT HANDLER FOR 1146

> BEGIN

> SELECT 'Table Books missing. Please create table first.' AS Message;

> END;

> INSERT INTO Books(BookID, Title) VALUES (inBookID, inTitle);

> SELECT count(\*) FROM Books WHERE BookID = inBookID;

> END $$

> DELIMITER ;

> CALL InsertBooks(1, 'Thrawn');

> CALL InsertBooks(1, 'Chaos Rising');

> DROP TABLE Books;

Text

Description automatically generated> CALL InsertBooks(2, 'Hello There');