Relational Databases with MySQL Week 4 Coding Assignment

Points possible: 70

Category	Criteria	% of Grade
Functionality	Does the code work?	25
Organization	Is the code clean and organized? Proper use of white space, syntax, and consistency are utilized. Names and comments are concise and clear.	25
Creativity	Student solved the problems presented in the assignment using creativity and out of the box thinking.	25
Completeness	All requirements of the assignment are complete.	25

Instructions: Using a text editor of your choice, write the queries that accomplishes the objectives listed below. Take screenshots of the queries and results and paste them in this document where instructed below. Create a new repository on GitHub for this week's assignments and push this document, with your Java project code, to the repository. Lastly, in the Learning Management System, click the "Add Submission" button and paste the URL to your GitHub repository.

Coding Steps:

Write 5 stored procedures for the employees database.

Write a description of what each stored procedure does and how to use it.

Procedures should use constructs you learned about from your research assignment and be more than just queries.

Screenshots:

1. First procedure is to calculate the average salary of an employee without showing their total salary. This will allow HR people to see what "bucket" they belong in, while keeping their total salary private.

```
DROP PROCEDURE IF EXISTS average_salary;

DELIMITER $$

CREATE PROCEDURE average_salary(IN employee_number int, OUT salary_range varchar(100))
BEGIN

    DECLARE avg_total_salary int;

    SELECT AVG(s.salary)
    INTO avg_total_salary
    FROM salaries s
    WHERE emp_no = employee_number;

CASE

WHEN avg_total_salary  >= 35000 AND avg_total_salary < 45000 THEN
    SET salary_range = 'under $45,000';
WHEN avg_total_salary >= 45000 AND avg_total_salary < 55000 THEN
    SET salary_range = 'between $45,000 and $55,000';
WHEN avg_total_salary >= 55000 AND avg_total_salary < 65000 THEN
    SET salary_range = 'between $55,000 and $65,000';
WHEN avg_total_salary >= 65000 AND avg_total_salary < 75000 THEN
    SET salary_range = 'between $65,000 and $75,000';
ELSE
    SET salary_range = 'over $75,000';
END CASE;

END $$

DELIMITER ;

CALL average_salary (10014, @salary);
SELECT @salary;
```

2. The second procedure is to show an HR person an employees birthday month, without disclosing the year they are born. If someone wants to create a calendar of birthdays, but now know how old they are, they can use this procedure and it will only tell them the month they are born.

```
DROP PROCEDURE if EXISTS employee_birthday_month;
DELIMITER $$
CREATE PROCEDURE employee_birthday_month(IN employee_number int, OUT birth_month varchar(100))
BEGIN
    DECLARE year_into_month int;

    SELECT MONTH(e.birth_date)
    INTO year_into_month
    FROM employees e
    WHERE emp_no = employee_number;

CASE
WHEN year_into_month = 1 THEN
    SET birth_month = 'January';
WHEN year_into_month = 2 THEN
    SET birth_month = 'February';
WHEN year_into_month = 3 THEN
    SET birth_month = 'March';
WHEN year_into_month = 4 THEN
    SET birth_month = 'April';
WHEN year_into_month = 5 THEN
    SET birth_month = 'May';
WHEN year_into_month = 6 THEN
    SET birth_month = 'June';
WHEN year_into_month = 7 THEN
    SET birth_month = 'July';
WHEN year_into_month = 8 THEN
    SET birth_month = 'August';
WHEN year_into_month = 9 THEN
    SET birth_month = 'September';
WHEN year_into_month = 10 THEN
    SET birth_month = 'October';
WHEN year_into_month = 11 THEN
    SET birth_month = 'November';
ELSE
    SET birth_month = 'December';
END CASE;

END $$

DELIMITER ;

CALL employee_birthday_month (10041, @birthday);
SELECT @birthday;
```

3. The third procedure will show someone how many titles/promotions an employee has had during their time at the company.

```
DROP PROCEDURE IF EXISTS number_of_titles;
DELIMITER $$
CREATE PROCEDURE number_of_titles(IN employee_number int, OUT title_numbers int)
BEGIN

    SELECT COUNT(*)
    INTO title_numbers
    FROM titles
    WHERE emp_no = employee_number;

END $$
DELIMITER ;

CALL number_of_titles (10009, @title);
SELECT @title;
```

4. The fourth procedure will show how many years an employee has been at the company.

```
DROP PROCEDURE IF EXISTS years_in_job;
DELIMITER $$

CREATE PROCEDURE years_in_job (IN employee_number int, OUT years_in_job int)
BEGIN
DECLARE start_year int;
DECLARE end_year int;

SELECT year(de.from_date), year(de.to_date)
INTO start_year, end_year
FROM dept_emp de
WHERE de.emp_no = employee_number
LIMIT 1;

IF end_year = 9999 THEN
SET end_year= year(now());
END IF;

SELECT end_year - start_year INTO years_in_job;
END $$
DELIMITER ;

CALL years_in_job (10009, @years);
SELECT @years;
```

5. The fifth procedure will show how many employees are in a department.

```
DROP PROCEDURE IF EXISTS employees_in_department;
DELIMITER $$

CREATE PROCEDURE employees_in_department (IN department_name varchar(100), OUT number_of_employees int)
BEGIN
    SELECT COUNT(emp_no)
    INTO number_of_employees
    FROM dept_emp de
    INNER JOIN departments d ON de.dept_no = d.dept_no
    WHERE dept_name = department_name;
END $$
DELIMITER ;
```

URL to GitHub Repository:

https://github.com/Christinalytle/week4SQLHomework.git