Assignment 3 – Solution:
Indexing and transaction management

Assignment description:

In this assignment you are expected to create indexing and transactions to improve query performance in your database. You are given the following instructions to answer the assignment:

1. Create two tables (relations) named Emp and Dept with the following schema:
   Emp (eid: integer, salary: integer, age: int, did: integer)
   Dept (did: integer, floor: integer, budget: real)

Solution1:

```
CREATE DATABASE DB;

CREATE TABLE `Emp` (

`eid` INT NOT NULL,

`salary` INT NOT NULL,

`age` INT NOT NULL,

`did` INT NOT NULL,

PRIMARY KEY (`eid`)

FOREIGN KEY (‘did’) REFERENCES Dept (‘did’)

);


CREATE TABLE `Dept` (

`did` INT NOT NULL auto_increament,

`floor` INT NOT NULL,

`budget` REAL NOT NULL,

PRIMARY KEY (`did`)

);
```

2. Generate 300,000 employee records according to the following:
   Generate random numbers to represent age, salary, floor and budget. The range of age should be between 18 and 60, for the salary between 15 and 30, for the floor between 1 and 5 and for the budget between 20 and 100.

   Note1: For simplicity in generating random values, all attribute types are considered as either integer or real values.
   Note2: 'eid' and 'did' can be primary keys and be automatically increased.

   - In the submission file provide the SQL codes used for generating the records.

Solution2:

```
DELIMITER $$
CREATE PROCEDURE InsertEmp(IN NumRows INT)
BEGIN
        DECLARE i INT DEFAULT 1;
        START TRANSACTION;
                WHILE i <= NumRows DO
                        INSERT INTO Emp(salary, age, did)
                        VALUES
                                (i, 15 + CEIL(RAND() * (30 - 15)),
                                 18 + CEIL(RAND() * (60 - 18)),
                                1+ CEIL(RAND() * (5-1 )));
                        SET i = i + 1;
                END WHILE;
        COMMIT;
END$$
DELIMITER ;
CALL InsertEmp(300000);
**********************************************
DELIMITER $$
CREATE PROCEDURE InsertDept(IN NumRows INT)
BEGIN
        DECLARE i INT DEFAULT 1;
```

```
        START TRANSACTION;
            WHILE i <= NumRows DO
                INSERT INTO Dept(FLOOR, BUDGET)
                 VALUES
                (i, 1 + CEIL(RAND() * (5 - 1)),
                20 + CEIL(RAND() * (100 - 20)));
                SET i = i + 1;
            END WHILE;
        COMMIT;
END$$
DELIMITER ;
CALL InsertDept(5);
```

3. Write SQL queries to answer the following questions:
   3.1. How many employees are 45 years old?
   3.2. How many employees receive salary grater that 40?
   3.3. How many employees work in each department?
   3.4. What is the average budget over all departments?
   3.5. How many employees who are between 33 and 45 years old and receive salary greater than 20.
   3.6. What is the maximum salary among employees in the fifth floor?

- In the submission file provide both queries and answers.

Solution3:

3.1

```
SELECT COUNT( * ) as "No. of 45 years old employees"
FROM Emp
WHERE age=45;
```

3.2

```
SELECT COUNT( * ) as "No. of employees salary above 40"
FROM Emp
WHERE salary >40;
```

3.3

```
SELECT did, COUNT(*)
FROM Emp
GROUP BY did;
```

3.4

```
SELECT did, AVG(budget)
FROM Dept
GROUP BY did;
```

3.5

```
SELECT COUNT( * )
FROM Emp
WHERE (age BETWEEN 35 AND 45) AND (salary > 20);
```

3.6

```
SELECT max(salary)
FROM Emp
WHERE ( SELECT did
        FROM Dept
        WHERE floor=5);
```

4. Use profiling command to compute the execution time of each query.

- In the submission file provide a summary of execution times of queries.
- Utilize indexing to optimize the queries and re-run the same queries. Now repeat the profiling and compare the results with the first part. Use multi-indexing where necessary.
- In the submission file provide the comparison results.
- For each query which was optimized by indexing, mention the attributes that should be used as indices to compute the answer.

- In the submission file write the attributes and their reference relation for each query. For example: E.eid for Query number 3.x)

Solution4:

If you are using MySQL database server, you can use ***set profiling = 1;*** command to start profiling and after you are finished running the queries, you can use ***show profiles;*** command to show the execution time of each query. If you use some sort of GUI application to manage your database, you can check if it shows the query execution time after running each query.

Creating indexes

3.1. CREATE INDEX employee_age on Emp (age);

This index would speed up the query in Question 3.1 much faster.

3.2. CREATE INDEX employee_salary on Emp (salary);

This index would speed up the query in Question 3.2 much faster.

3.3 CREATE INDEX employee_did on Emp (did);

This index would speed up the query in Question 3.3 much faster.

3.4 CREATE INDEX department_budget on Dept (budget);

This index would speed up the query in Question 3.4 a bit faster.

3.5. CREATE INDEX employee_age_salary on Emp (age, salary);

This index would speed up the query in Question 3.5 a bit faster.

3.6. CREATE INDEX employee_did_salary on Emp (did, salary);

This index would speed up the query in question 3.6 a bit faster.

5. For each of the following transactions write a procedure and call it (with optional values if needed):
   5.1. Recruit an employee identified by 'eid' in the department in floor 2. (Insert new employee)
   5.2. Reduce the budget of the departments identified by 'did' in the first floor by 20 percent.
   5.3. Increase the salary of the employees who work in the departments with budget greater than 80, by ten percent.

- In the submission file provide the procedure codes and sample results.

Solution5:

5.1 .

```
CREATE PROCEDURE HireEmployee (INT eid,IN salary INT, IN age INT , IN did INT)

BEGIN

    INSERT INTO Emp( eid, salary, age, did)

    VALUES(eid, salary,age,did);

END$$
```

Before calling the procedure, we find a department 'id' existing in the second floor by:

*SELECT * FROM Dept WHERE floor=2;*

CALL HireEmployee (500000, 30, 39, **1**)

Note: In this CALL We assume that **did=1** is in the second floor. (for your database this number can be different)

5.2

```
CREATE PROCEDURE ReduceSalary(IN decrease_r FLOAT,IN floor_n INT)
    BEGIN
        UPDATE Dept
        SET budget = budget*(1-decrease_r )
        WHERE floor = floor_n;
    END$$
CALL ReduceSalary (0.2,1)
```

5.3

```
CREATE PROCEDURE IncreaseSalary(IN increase_r FLOAT ,IN dept_budget FLOAT)
    BEGIN
    START TRANSACTION;
    UPDATE Emp
    INNER JOIN Dept
        ON Dept.did = EMP.did
    SET Emp.salary = Emp.salary*(1+increase_r)
    WHERE Dept.budget > dept_budget;
    COMMIT;
    END$$
CALL IncreaseSalary(0.1, 80.0);
```