## Indian Institute of Information Technology Surat

****

# Lab Report on

# Advanced Database Management (CS 604) Practical

**Submitted by**

### [RAHUL KUMAR SINGH] (UI21CS44)

**Course Faculty**

### Mr. Rishi Sharma

## Department of Computer Science and Engineering

## Indian Institute of Information Technology Surat

## Gujarat-394190, India

**Jan-2024**

## Lab No: 5

**Aim: To implement Deadlock Detection Algorithm for Distributed Database using Wait-for Graph to check for Deadlock.**

**Description:**

* DetectDeadlock Procedure:
* Creates a temporary table for wait-for graph.
* Populates wait-for graph with data from deadlock\_info.
* DepthFirstSearch Procedure:
* Simulates stack for DFS using a temporary table.
* Detects cycles by recursively traversing wait-for graph.
* DetectDeadlock Procedure Modification:
* Inserts multiple rows into dfs\_stack from wait\_for\_graph where requesting\_node=start\_node.

## Source Code:

**Table:**

CREATE TABLE Deadlock\_Info (

transaction\_id INT PRIMARY KEY AUTO\_INCREMENT,

requesting\_node INT,

holding\_node INT

);

**Detect Deadlock Procedure:**

DELIMITER //

CREATE PROCEDURE DetectDeadlock()

BEGIN

DECLARE result INT DEFAULT 0;

DECLARE temp INT DEFAULT 0;

DECLARE start\_node INT;

DECLARE current\_node INT;

DECLARE done INT DEFAULT 0;

CREATE TEMPORARY TABLE IF NOT EXISTS wait\_for\_graph (

requesting INT,

holding INT

);

INSERT INTO wait\_for\_graph SELECT requesting\_node, holding\_node FROM deadlock\_info;

SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

CREATE TEMPORARY TABLE IF NOT EXISTS distinct\_nodes (

node INT

);

INSERT INTO distinct\_nodes

SELECT DISTINCT requesting

FROM wait\_for\_graph;

WHILE (SELECT COUNT(\*) FROM distinct\_nodes) > 0 DO

SELECT node INTO start\_node FROM distinct\_nodes ORDER BY node LIMIT 1;

DELETE FROM distinct\_nodes WHERE node = start\_node;

SELECT CONCAT("Start Node: ",start\_node) as message;

SELECT result;

CALL DepthFirstSearch(start\_node, start\_node, temp);

IF temp = 1 THEN

SET result = 1;

END IF;

DELETE FROM wait\_for\_graph;

INSERT INTO wait\_for\_graph SELECT requesting\_node, holding\_node FROM deadlock\_info;

END WHILE;

SELECT result;

IF result = 0 THEN

SELECT "No Deadlock Detected!" as message;

ELSE

SELECT "Deadlock Detected!" as message;

END IF;

DROP TEMPORARY TABLE IF EXISTS wait\_for\_graph;

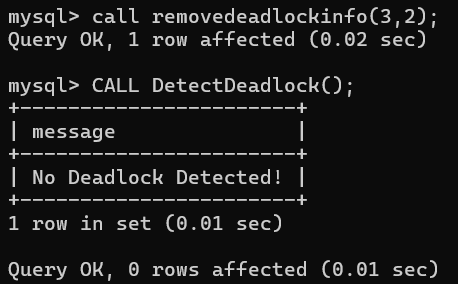
DROP TEMPORARY TABLE IF EXISTS distinct\_nodes;

END //

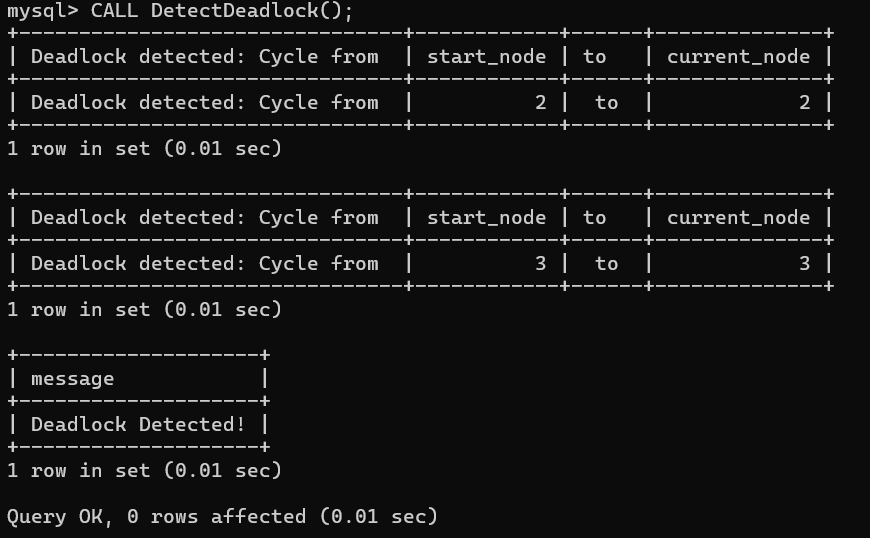
DELIMITER ;

## Output:

**In case of No Deadlock:**

****

**In case of Deadlock:**

****

## Conclusion:

* Detects deadlocks in a distributed database using a wait-for graph.
* Implemented depth-first search within MySQL stored procedures.
* Procedures manage deadlock information, simulate DFS, and execute deadlock detection.
* MySQL limitations for complex algorithms; use external languages for efficiency.
* Deadlock detection results in cycles which means careful consideration using it.

.