**MSE**

**AVERAGE LEARNER**

**ACTIVITY**

1. Implement a procedure to demonstrate deadlock occurring between transactions.

Deadlock occurs when two or more transactions are waiting for each other to give up on lock in an infinite manner. Deadlock brings the whole database management system on halt.

Example:

Suppose, Transaction T1 holds a lock on some rows in the Employee table and needs to update some rows in the Salary table. Simultaneously, Transaction T2 holds locks on those very rows which T1 needs to update in the Salary table but needs to update the rows in the Employee table held by Transaction T1.

Now, the main problem comes, Transaction T1 will wait for transaction T2 to give up lock, and similarly transaction T2 will wait for transaction T1 to give up lock. As a consequence, All activity comes to a halt and remains at a standstill forever unless the DBMS detects the deadlock and aborts one of the transactions.

**Avoiding Deadlocks:**

Using both row-level locking and the TRANSACTION\_READ\_COMMITTED isolation

level makes it likely that you will avoid deadlocks.

However, deadlocks are still possible.

Developers can avoid deadlocks by using consistent application logic:

for example, transactions that access Employee and Salary should always access the

tables in the same order. That way, in the scenario described above, Transaction T2

simply waits for transaction T1 to release the lock on Salary before it begins. When

transaction T1 releases the lock on Salary, Transaction T2 can proceed freely.

Another tool available to you is the LOCK TABLE statement. A transaction can attempt

to lock a table in exclusive mode when it starts to prevent other transactions from getting

shared locks on a table.