ADVANCED TOPICS IN DATABASE SYSTEMS

Database Isolation Levels Simulation Program

Prepared by Harun Yahya ÜNAL

19/04/2025

This project is the first term project of the SE 308 – Advanced Topics in Database Systems course. The goal of the project is to simulate user transactions on the AdventureWorks2022 database and measure the performance under different **transaction isolation levels**.

There are two types of users in the simulation:

- **Type A users** make update queries.
- **Type B users** make select queries.

The simulation program is built with multi-threading by using C# and Windows Forms. Each user runs the same query structure 100 times. During the simulation, the program measures the time for each thread and counts the number of deadlocks.

The tests are done in two different cases:

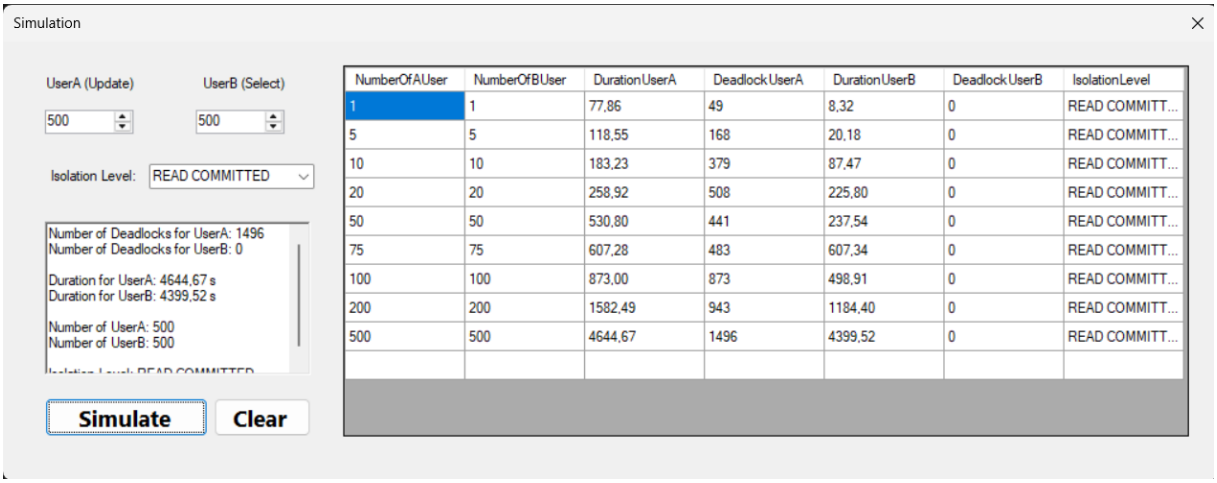
- **Part 1:** Without indexes on SalesOrder tables.
- **Part 2:** With indexes on SalesOrder tables to improve performance.

In this report, the source codes of the simulation program, test results, and screenshots are shared clearly. All results are compared for each isolation level.

Part 1 (Without Indexes)

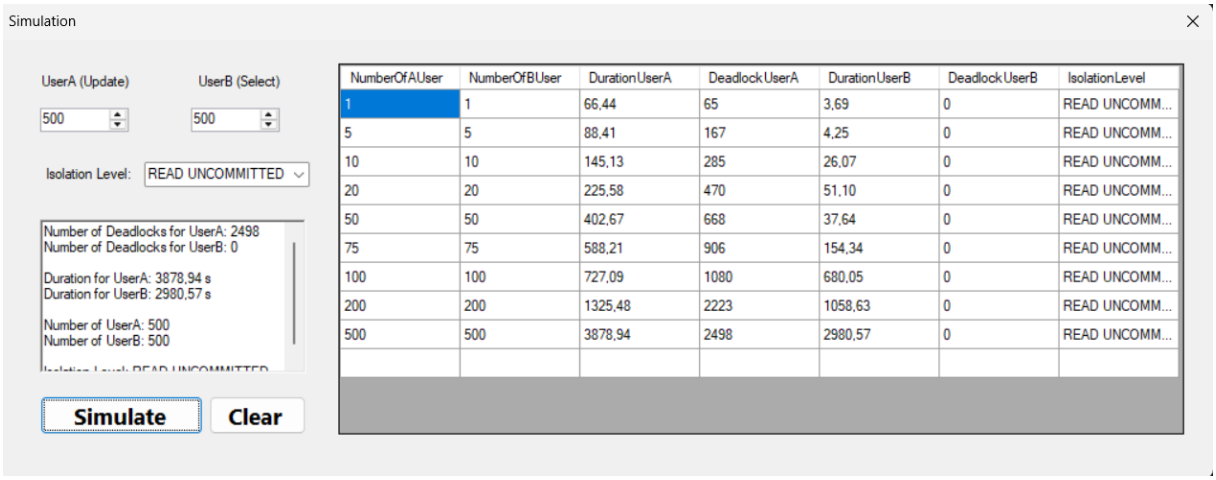
- READ COMMITTED

Isolation Level	READ COMMITTED				
Number of User A	Number of User B	Duration of Type A Threads (s)	Number of Deadlocks of User A	Duration of Type B Threads (s)	Number of Deadlocks of User B
1	1	77,86	49	8,32	0
5	5	118,55	168	20,18	0
10	10	183,23	379	87,47	0
20	20	258,92	508	225,80	0
50	50	530,80	441	237,54	0
75	75	607,28	483	607,34	0
100	100	873,00	873	498,91	0
200	200	1582,49	943	1184,40	0
500	500	4644,67	1496	4399,52	0



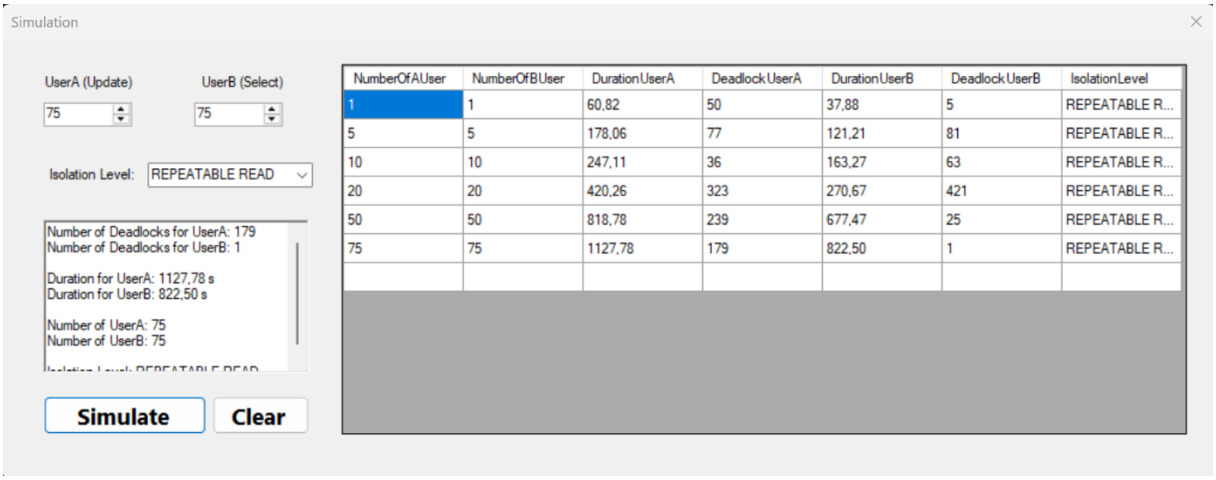
- READ UNCOMMITED

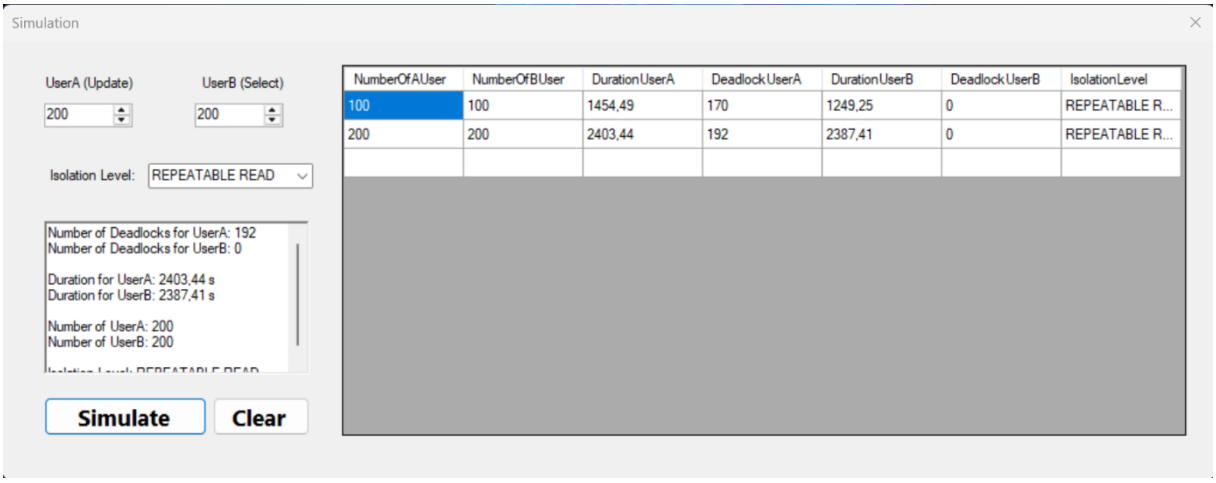
Isolation Level	READ UNCOMMITTED				
Number of User A	Number of User B	Duration of Type A Threads (s)	Number of Deadlocks of User A	Duration of Type B Threads (s)	Number of Deadlocks of User B
1	1	66,44	66	3,69	0
5	5	88,41	167	4,25	0
10	10	145,13	285	26,07	0
20	20	225,58	470	51,10	0
50	50	402,67	668	37,64	0
75	75	588,21	906	154,34	0
100	100	727,09	1080	680,05	0
200	200	1325,48	2223	1058,63	0
500	500	3878,94	2498	2980,57	0



- REPEATABLE READ

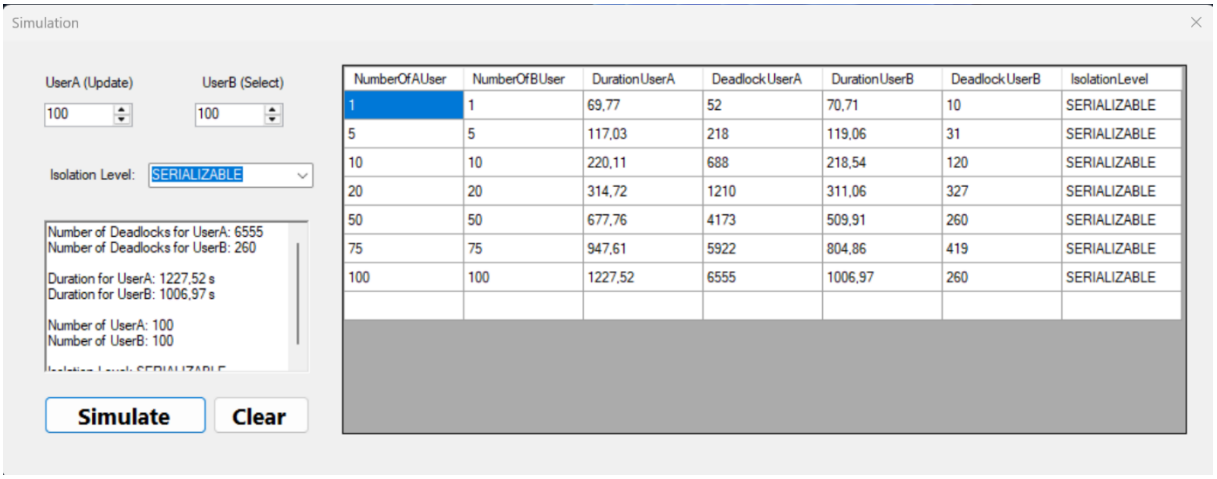
Isolation Level	REPEATABLE READ				
Number of User A	Number of User B	Duration of Type A Threads (s)	Number of Deadlocks of User A	Duration of Type B Threads (s)	Number of Deadlocks of User B
1	1	60,82	50	37,88	5
5	5	178,06	77	121,21	81
10	10	247,11	36	163,27	63
20	20	420,26	323	270,67	421
50	50	818,78	239	677,47	25
75	75	1127,78	179	822,50	1
100	100	1454,49	170	1249,25	0
200	200	2403,44	192	2387,41	0

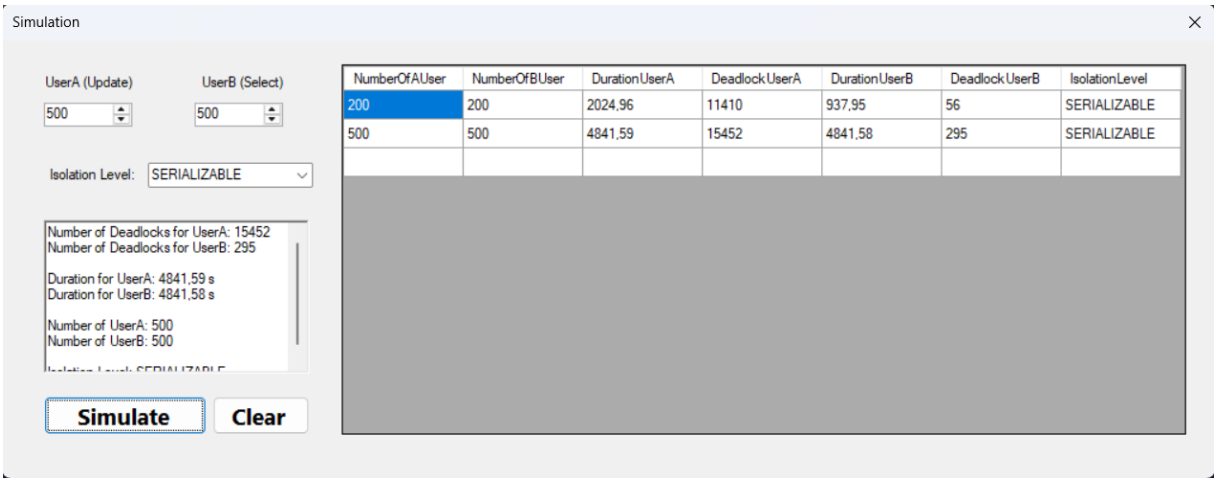




- SERIALIZABLE

Isolation Level	SERIALIZABLE				
Number of User A	Number of User B	Duration of Type A Threads (s)	Number of Deadlocks of User A	Duration of Type B Threads (s)	Number of Deadlocks of User B
1	1	69,77	52	70,71	10
5	5	117,03	218	119,06	31
10	10	220,11	688	218,54	120
20	20	314,72	1210	311,06	327
50	50	677,76	4173	509,91	260
75	75	947,61	5922	804,86	419
100	100	1227,52	6555	1006,97	260
200	200	2024,96	11410	937,95	56
500	500	4841,59	15452	4841,58	295





Part 2 (With Indexes)

Used Index for **SalesOrderHeader** table:

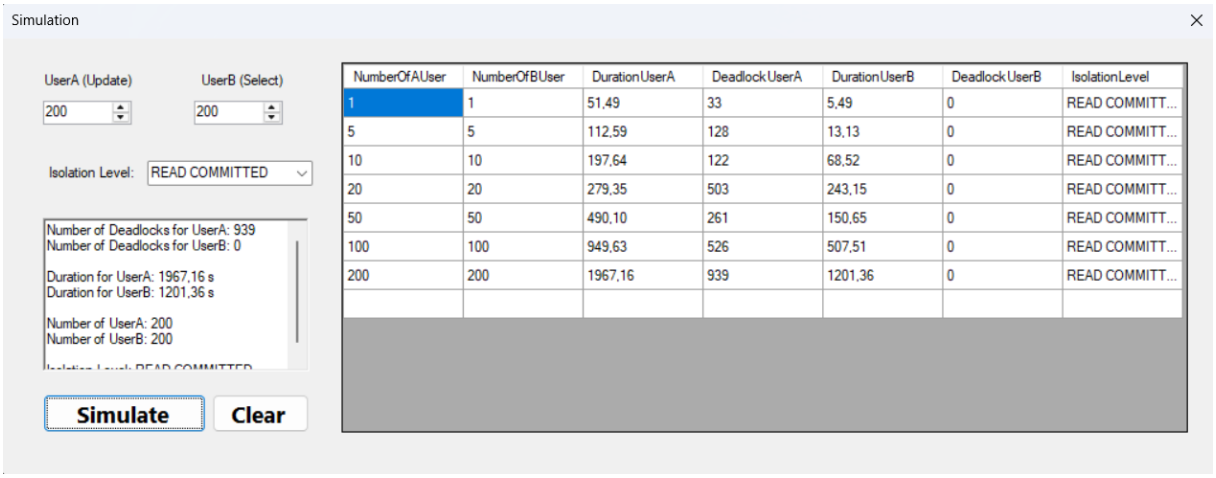
```
CREATE NONCLUSTERED INDEX IDX_OrderDate_OnlineOrderFlag_SalesOrderID
ON Sales.SalesOrderHeader (OrderDate, OnlineOrderFlag)
INCLUDE (SalesOrderID);
```

Used Index for **SalesOrderDetail** table:

```
CREATE NONCLUSTERED INDEX IDX_SalesOrderID_UnitPrice
ON Sales.SalesOrderDetail (SalesOrderID)
INCLUDE (UnitPrice);
```

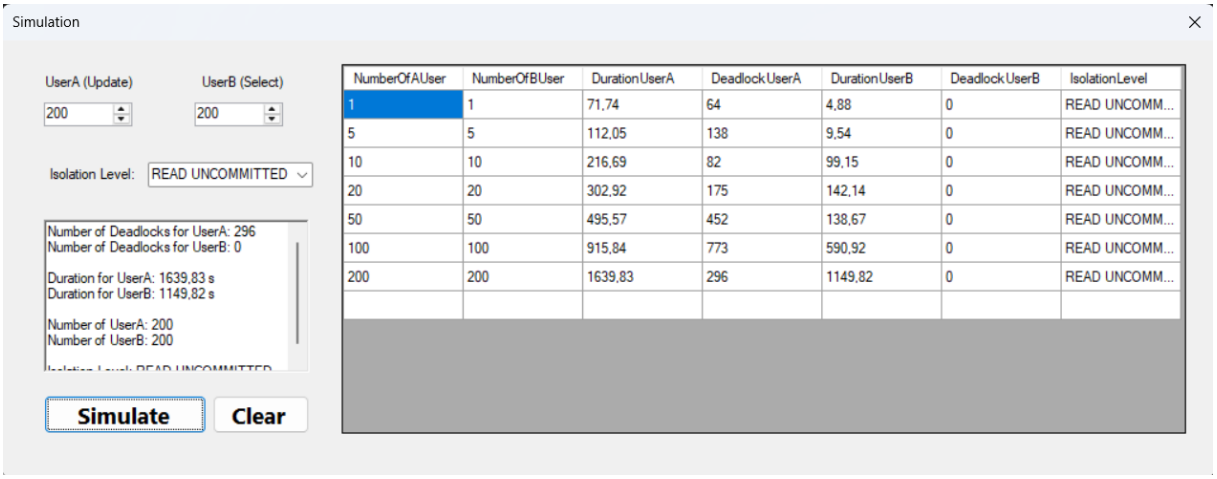
- READ COMMITTED

Isolation Level	READ COMMITTED				
Number of User A	Number of User B	Duration of Type A Threads (s)	Number of Deadlocks of User A	Duration of Type B Threads (s)	Number of Deadlocks of User B
1	1	51,49	33	5,49	0
5	5	112,59	128	13,13	0
10	10	197,64	122	68,52	0
20	20	279,35	503	243,15	0
50	50	490,10	261	150,65	0
100	100	949,63	526	507,51	0
200	200	1967,16	939	1201,36	0



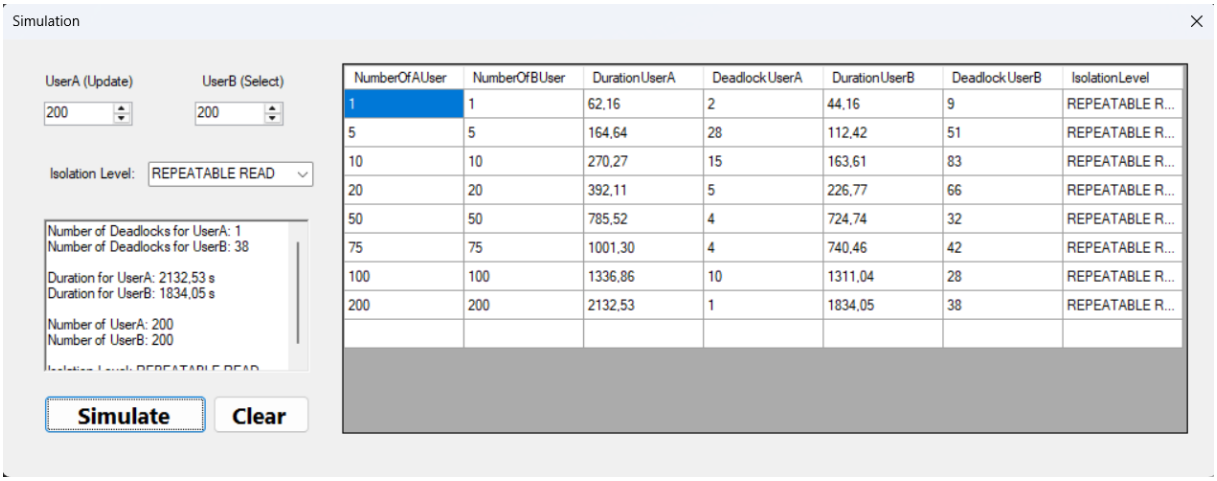
- READ UNCOMMITED

Isolation Level	READ UNCOMMITTED				
Number of User A	Number of User B	Duration of Type A Threads (s)	Number of Deadlocks of User A	Duration of Type B Threads (s)	Number of Deadlocks of User B
1	1	71,74	64	4,88	0
5	5	112,05	138	9,54	0
10	10	216,69	82	99,15	0
20	20	302,92	175	142,14	0
50	50	495,57	452	138,67	0
100	100	915,84	773	590,92	0
200	200	1639,83	296	1149,82	0



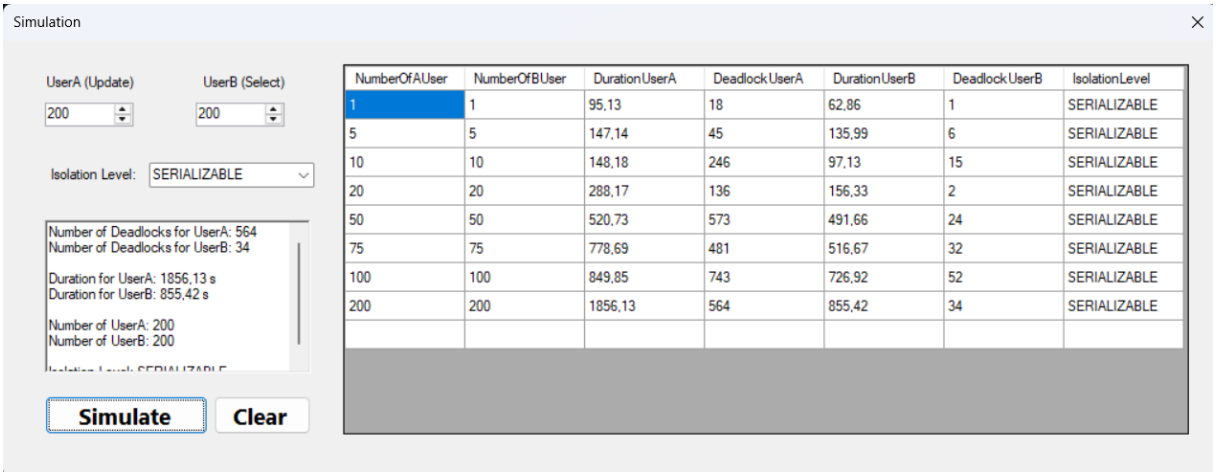
- REPEATABLE READ

Isolation Level	REPEATABLE READ				
Number of User A	Number of User B	Duration of Type A Threads (s)	Number of Deadlocks of User A	Duration of Type B Threads (s)	Number of Deadlocks of User B
1	1	62,16	2	44,16	9
5	5	164,64	28	112,42	51
10	10	270,27	15	163,61	83
20	20	392,11	5	226,77	66
50	50	785,52	4	724,74	32
75	75	1001,30	4	740,46	42
100	100	1336,86	10	1311,04	28
200	200	2132,53	1	1834,05	38



- SERIALIZABLE

Isolation Level	SERIALIZABLE				
Number of User A	Number of User B	Duration of Type A Threads (s)	Number of Deadlocks of User A	Duration of Type B Threads (s)	Number of Deadlocks of User B
1	1	95,13	18	62,86	1
5	5	147,14	45	135,99	6
10	10	148,18	246	97,13	15
20	20	288,17	136	156,33	2
50	50	520,73	573	491,66	24
75	75	778,69	481	516,67	32
100	100	849,85	743	726,92	52
200	200	1856,13	564	855,42	34



Source Codes

DatabaseOperations.cs

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Threading.Tasks;

namespace Advanced_Database_Topics
{
  public class DatabaseOperations
  {
    private readonly Simulation _simulation;
    private readonly string _connectionString;
```

```
    public DatabaseOperations(Simulation simulation)
    {
        _connectionString = "Data Source=localhost\\SQLEXPRESS;Initial
Catalog=AdventureWorks2022;Integrated
Security=True;MultipleActiveResultSets=True;Connect Timeout=120;Max Pool
Size=100000;";
        _simulation = simulation;
    }

    public async Task GetDataAsync(string isolationLevel)
    {
        //User B
        List<Task> sqlCommadTasks = new List<Task>();

        for (int i = 0; i < 100; i++)
        {
            int j = i;
            sqlCommadTasks.Add(Task.Run(() => SelectOperation(isolationLevel, j)));
        }
        try
        {
            await Task.WhenAll(sqlCommadTasks);
        }
        catch (Exception)
        {

            throw;
        }
        sqlCommadTasks.Clear();
    }

    private void SelectOperation(string isolationLevel, int i)
    {
        Random random = new Random();

        using (SqlConnection connection = new SqlConnection(_connectionString))
        {
            connection.Open();

            using (SqlTransaction transaction = connection.BeginTransaction())
```

```
        {

            try
            {
                SetIsolationLevel(connection, transaction, isolationLevel);

                if (random.NextDouble() < 0.5)
                {
                    SqlCommand cmd = new SqlCommand(SelectQuery("20110101",
"20111231"), connection, transaction);
                    cmd.ExecuteNonQuery();
                    Console.WriteLine("User B: " + i);
                }
                if (random.NextDouble() < 0.5)
                {
                    SqlCommand cmd = new SqlCommand(SelectQuery("20120101",
"20121231"), connection, transaction);
                    cmd.ExecuteNonQuery();
                    Console.WriteLine("User B: " + i);
                }
                if (random.NextDouble() < 0.5)
                {
                    SqlCommand cmd = new SqlCommand(SelectQuery("20130101",
"20131231"), connection, transaction);
                    cmd.ExecuteNonQuery();
                    Console.WriteLine("User B: " + i);
                }
                if (random.NextDouble() < 0.5)
                {
                    SqlCommand cmd = new SqlCommand(SelectQuery("20140101",
"20141231"), connection, transaction);
                    cmd.ExecuteNonQuery();
                    Console.WriteLine("User B: " + i);
                }
                //if (random.NextDouble() < 0.5)
                //{
                //    SqlCommand cmd = new SqlCommand(SelectQuery("20150101",
"20151231"), connection, transaction);
                //    await cmd.ExecuteScalarAsync();
                //}
                // there is no record for 20150101 to 20151231 in the database
```

```
            transaction.Commit();
        }
        catch (SqlException ex) when (ex.Number == 1205) // deadlock error number
        {
            transaction.Rollback();
            _simulation.NumberOfDeadlocksUserB++;
            Console.WriteLine("Deadlock occurred for User B: " + i);
        }
        catch (Exception ex)
        {
            if (transaction.Connection != null)
            {
                transaction.Rollback();
            }
            Console.WriteLine("Error: " + ex.Message);
            return;
        }
      }
      connection.Close();
    }
}

public async Task UpdateDataAsync(string isolationLevel)
{
    //User A
    List<Task> sqlCommadTasks = new List<Task>();

    for (int i = 0; i < 100; i++)
    {
      int j = i;
      sqlCommadTasks.Add(Task.Run(() => UpdateOperation(isolationLevel, j)));
    }
    try
    {
      await Task.WhenAll(sqlCommadTasks);
    }
    catch (Exception)
    {

      throw;
    }
```

```
    }

    private void UpdateOperation(string isolationLevel, int i)
    {
      Random random = new Random();

      using (SqlConnection connection = new SqlConnection(_connectionString))
      {
        connection.Open();

        using (SqlTransaction transaction = connection.BeginTransaction())
        {

          try
          {
            SetIsolationLevel(connection, transaction, isolationLevel);

            if (random.NextDouble() < 0.5)
            {
              SqlCommand cmd = new SqlCommand(UpdateQuery("20110101",
"20111231"), connection, transaction);
              cmd.ExecuteNonQuery();
              Console.WriteLine("User A: " + i);
            }
            if (random.NextDouble() < 0.5)
            {
              SqlCommand cmd = new SqlCommand(UpdateQuery("20120101",
"20121231"), connection, transaction);
              cmd.ExecuteNonQuery();
              Console.WriteLine("User A: " + i);
            }
            if (random.NextDouble() < 0.5)
            {
              SqlCommand cmd = new SqlCommand(UpdateQuery("20130101",
"20131231"), connection, transaction);
              cmd.ExecuteNonQuery();
              Console.WriteLine("User A: " + i);
            }
            if (random.NextDouble() < 0.5)
            {
```

```
                SqlCommand cmd = new SqlCommand(UpdateQuery("20140101",
"20141231"), connection, transaction);
                cmd.ExecuteNonQuery();
                Console.WriteLine("User A: " + i);
              }
              //if (random.NextDouble() < 0.5)
              //{
              //   SqlCommand cmd = new SqlCommand(UpdateQuery("20150101",
"20151231"), connection, transaction);
              //   await cmd.ExecuteNonQueryAsync();
              //}
              // there is no record for 20150101 to 20151231 in the database
              transaction.Commit();
            }
            catch (SqlException ex) when (ex.Number == 1205) // deadlock error number
            {
              transaction.Rollback();
              _simulation.NumberOfDeadlocksUserA++;
              Console.WriteLine("Deadlock occurred for User A: " + i);
            }
            catch (Exception ex)
            {
              if (transaction.Connection != null)
              {
                transaction.Rollback();
              }
              Console.WriteLine("Error: " + ex.Message);
            }
          }
          connection.Close();
        }
      }

      private string SelectQuery(string begin, string end)
      {
        return $"SELECT SUM(Sales.SalesOrderDetail.OrderQty) " +
            $"FROM Sales.SalesOrderDetail " +
            $"WHERE UnitPrice > 100 " +
            $"AND EXISTS (SELECT * FROM Sales.SalesOrderHeader " +
                  $"WHERE Sales.SalesOrderHeader.SalesOrderID =
Sales.SalesOrderDetail.SalesOrderID " +
```

```
                $"AND Sales.SalesOrderHeader.OrderDate BETWEEN '{begin}' AND
'{end}' " +
                $"AND Sales.SalesOrderHeader.OnlineOrderFlag = 1)";
    }

    private string UpdateQuery(string begin, string end)
    {
      return $"UPDATE Sales.SalesOrderDetail " +
          $"SET UnitPrice = UnitPrice * 10.0 / 10.0 " +
          $"WHERE UnitPrice > 100 " +
          $"AND EXISTS (SELECT * FROM Sales.SalesOrderHeader " +
                $"WHERE Sales.SalesOrderHeader.SalesOrderID =
Sales.SalesOrderDetail.SalesOrderID " +
                $"AND Sales.SalesOrderHeader.OrderDate BETWEEN '{begin}' AND
'{end}' " +
                $"AND Sales.SalesOrderHeader.OnlineOrderFlag = 1)";
    }

    private void SetIsolationLevel(SqlConnection connection, SqlTransaction
transaction, string level)
    {
      SqlCommand cmd = new SqlCommand($"SET TRANSACTION ISOLATION LEVEL
{level};", connection, transaction);
      cmd.ExecuteNonQuery();
    }
  }
}
```

Simulation.cs

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Diagnostics;
using System.Threading.Tasks;

namespace Advanced_Database_Topics
{
  public class Simulation
  {
    public string DurationUserA { get; set; }
```

```
public string DurationUserB { get; set; }
public int NumberOfDeadlocksUserA { get; set; } = 0;
public int NumberOfDeadlocksUserB { get; set; } = 0;

private int _userA;
private int _userB;
private string _isolationLevel;
private DatabaseOperations dbOps;

public Simulation(int userA, int userB, string isolationLevel)
{
  _userA = userA;
  _userB = userB;
  dbOps = new DatabaseOperations(this);
  _isolationLevel = isolationLevel;
}

public async Task Simulate()
{
  await Task.WhenAll(UserA(), UserB());
}

private async Task UserA()
{
  Stopwatch stopwatchUserA = new Stopwatch();
  List<Task> tasksA = new List<Task>();

  stopwatchUserA.Start();

  //Creating UserA
  for (int i = 0; i < _userA; i++)
  {
    tasksA.Add(Task.Run(async () =>
    {
      try
      {
        await dbOps.UpdateDataAsync(_isolationLevel);
      }
      catch (Exception)
      {
        throw;
```

```
            }
          }));
        }
        await Task.WhenAll(tasksA);
        stopwatchUserA.Stop();
        DurationUserA = stopwatchUserA.Elapsed.TotalSeconds.ToString("F2");
      }

      private async Task UserB()
      {
        Stopwatch stopwatchUserB = new Stopwatch();
        List<Task> tasksB = new List<Task>();

        //Creating UserB
        stopwatchUserB.Start();
        for (int i = 0; i < _userB; i++)
        {
          tasksB.Add(Task.Run(async () =>
          {
            try
            {
              await dbOps.GetDataAsync(_isolationLevel);
            }
            catch (Exception)
            {
              throw;
            }
          }));

        }
        await Task.WhenAll(tasksB);
        stopwatchUserB.Stop();
        DurationUserB = stopwatchUserB.Elapsed.TotalSeconds.ToString("F2");
      }
    }
}
```

Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
```

```
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Advanced_Database_Topics
{
  internal partial class Form1 : Form
  {
    private BindingList<SimulationModel> _simulationModels;
    public Form1()
    {
      InitializeComponent();
      _simulationModels = new BindingList<SimulationModel>();
      dataGridView1.DataSource = _simulationModels;
    }

    private void Form1_Load(object sender, EventArgs e)
    {
      LoadIsolationLevelData();
    }

    private void LoadIsolationLevelData()
    {
      string[] isolationLevels = new string[] { "READ UNCOMMITTED", "READ
COMMITTED", "REPEATABLE READ", "SERIALIZABLE"};
      cbxIsolationLvl.DataSource = isolationLevels;
    }

    private async void btnSimulate_ClickAsync(object sender, EventArgs e)
    {
      richTextBox1.Clear();
      richTextBox1.Text = "Simulating...\n\n";
      await SimulateAsync();
    }

    private async Task SimulateAsync()
    {
      var numOfUserA = Convert.ToInt32(numUserA.Value);
      var numOfUserB = Convert.ToInt32(numUserB.Value);
      string isolationLevel = cbxIsolationLvl.SelectedItem.ToString();
```

```
        Simulation simulation = new Simulation(numOfUserA, numOfUserB,
isolationLevel);
        await simulation.Simulate();

        var newSimulation = new SimulationModel
        {
          NumberOfAUser = numOfUserA,
          NumberOfBUser = numOfUserB,
          DurationUserA = simulation.DurationUserA,
          DeadlockUserA = simulation.NumberOfDeadlocksUserA,
          DurationUserB = simulation.DurationUserB,
          DeadlockUserB = simulation.NumberOfDeadlocksUserB,
          IsolationLevel = isolationLevel
        };

        _simulationModels.Add(newSimulation);

        Console.WriteLine("**********The simulation is over.**********");
        richTextBox1.Text = $"Number of Deadlocks for UserA:
{simulation.NumberOfDeadlocksUserA}\nNumber of Deadlocks for UserB:
{simulation.NumberOfDeadlocksUserB}";
        richTextBox1.Text += $"\n\nDuration for UserA: {simulation.DurationUserA}
s\nDuration for UserB: {simulation.DurationUserB} s";
        richTextBox1.Text += $"\n\nNumber of UserA: {numOfUserA}\nNumber of UserB:
{numOfUserB}";
        richTextBox1.Text += $"\n\nIsolation Level: {isolationLevel}";
      }

      private void btnClear_Click(object sender, EventArgs e)
      {
        _simulationModels.Clear();
      }
    }
}
```