SE 308
Advanced Topics in Database Systems

Term Project 1

Işıl Deniz Öztürk
Hilal Ayışığı

A database transaction, in order to maintain consistency in a database must satisfy and follow the ACID properties. Sometimes,we want to allow concurrent users to run their transactions independently without interrupting the other transactions that are concurrently processing.A transaction acquires a lock prior to data access; the lock is released when the transaction is completed, records are committed or rollback so that another transaction can lock the data item for its own use. A shared Lock(Read lock), is issued when a transaction wants to read data no exclusive lock is held on that data item. An exclusive Lock(Write lock) is issued when a transaction wants to update a data item and no locks are currently held on that data item by any other transaction.

Isolation is one of the properties from ACID of a database transaction which determines how two transactions are isolated from each other.For that we have various isolation levels defined in Sql server. Degree of isolation on transaction means how two transactions will be isolated to and to what extent. Isolation levels determine when and in what case the records will be locked, as well as how isolated transactions will look from each other.

Transactions specify an isolation level that defines the degree to which one transaction must be isolated from resource or data modifications made by other transactions.
Transaction isolation levels control, whether locks are taken when data is read, and what type of locks are requested. How long the read locks are held.

The lowest isolation level is read uncommitted.Level that does not lock any records and is not affected by any locks. In this level transaction can read data inserted,updated or deleted by another transaction which is not committed, thus allowing dirty reads to happen which happens when a transaction reads data modified by another concurrent transaction that has not been committed yet. We might end up using incorrect data because we don't know if that other transaction will be committed or rollback. Also data can be changed between individual statements by another transaction within the current transaction which causes non repeatable reads or phantom reads.

Read committed, default level of sql server. Transactions cannot read any records/changes that have not been committed by another transaction, can only see data that has been committed so dirty read is no longer possible. However non repeatable reads which means when trying to select the row again with the same transaction, the record that we just read cannot be guaranteed this time, can receive different result and phantom reads, the record we just read may have changed, someone may have inserted/updated/deleted it while we were reading these records may appear as a phantom record so they are still possible.

Repeatable read, which is a bit more strict level, ensures that the same select query will always return the same result, no matter how many times it is executed. At the beginning of the transaction we read some records and put shared locks on these records so that no one else would change them. So when we read these records again, we know that we will continue to see the same record because no one changed it. However if another transaction inserts a new record that matches the condition that we read, we couldn't put a lock on it because we hadn't seen that record before and so that now we can see these inserted records in our 2nd reading.

Serializable, at its highest level, isolation ensures that all concurrent transactions will not affect each other. Concurrent transactions running in this level are guaranteed to be able to yield the same result as if they’re executed in some order, one after another without overlapping. When we put a lock on the records, while being in this level, others cannot even insert a record that matches the condition that we used in our query.

Isolation Level	**READ UNCOMMITTED**				
Number of Type A Users	Number of Type B Users	Average Time of Type A Threads (second)	Number of Deadlocks Encountered by Type A Users	Average Time of Type B Threads (second)	Number of Deadlocks Encountered by Type B Users
1	1	9,8	0	3,2	0
2	2	42,1	70	11,4	0
3	3	36,4	144	3,5	0
4	4	37,7	227	4,3	0
5	5	46,8	246	6,8	0
6	6	59,3	386	5	0
7	7	54,2	398	6,3	0
8	8	77.4	544	7,0	0
9	9	84,1	581	7,0	0
10	10	79,4	588	9,7	0
25	25	869,9	7703	207,0	1598
40	40	833,1	7766	173,8	32
65	65	398,0	4182	74.8	0
80	80	607,0	5391	130.7	0

As the number of users increases, the deadlocks increase, and the time gets longer. However, you cannot see deadlocks in B type users due to the READ UNCOMMITTED dirty read feature, because the server puts a share lock and we can do a dirty read with share lock.At this level, when we make a select from this table and read records, Thread B can read dirty read with select query, since it is not affected if there are locks in the records in the table, so the deadlock numbers are in the same way as in the table.

Isolation Level	**READ COMMITTED**				
Number of Type A Users	Number of Type B Users	Average Time of Type A Threads (second)	Number of Deadlocks Encountered by Type A Users	Average Time of Type B Threads (second)	Number of Deadlocks Encountered by Type B Users
1	1	18,3	0	17,3	0
2	2	44,0	62	46,6	10
3	3	51,5	161	56,1	70
4	4	50,3	192	40,9	30
5	5	86,5	290	77,9	171
6	6	131,6	374	91,0	133
7	7	134,0	451	120,6	100
8	8	138,8	536	158,4	237
9	9	170,3	539	176,6	308
10	10	130,5	643	105,8	244
25	25	448,5	1982	328,9	946
40	40	601,9	2899	484,2	1354
65	65	721,2	4246	477,7	2092
80	80	873,5	4776	658,2	3616

At this level, thread A was able to update those records, even though thread B had locked the records when selecting. Thread A Locks records to update, when thread B wants to lock them to read, it cannot read them until Thread A commits or rollback those changes, so dirty read is not possible at this level. As the number of users who want to access these records increases simultaneously with the circulation, the number of deadlocks also increases with the average time of threads. Unlike the read uncommitted, because B thread cannot do at this level, its deadlock numbers also increase as the number of users increases.

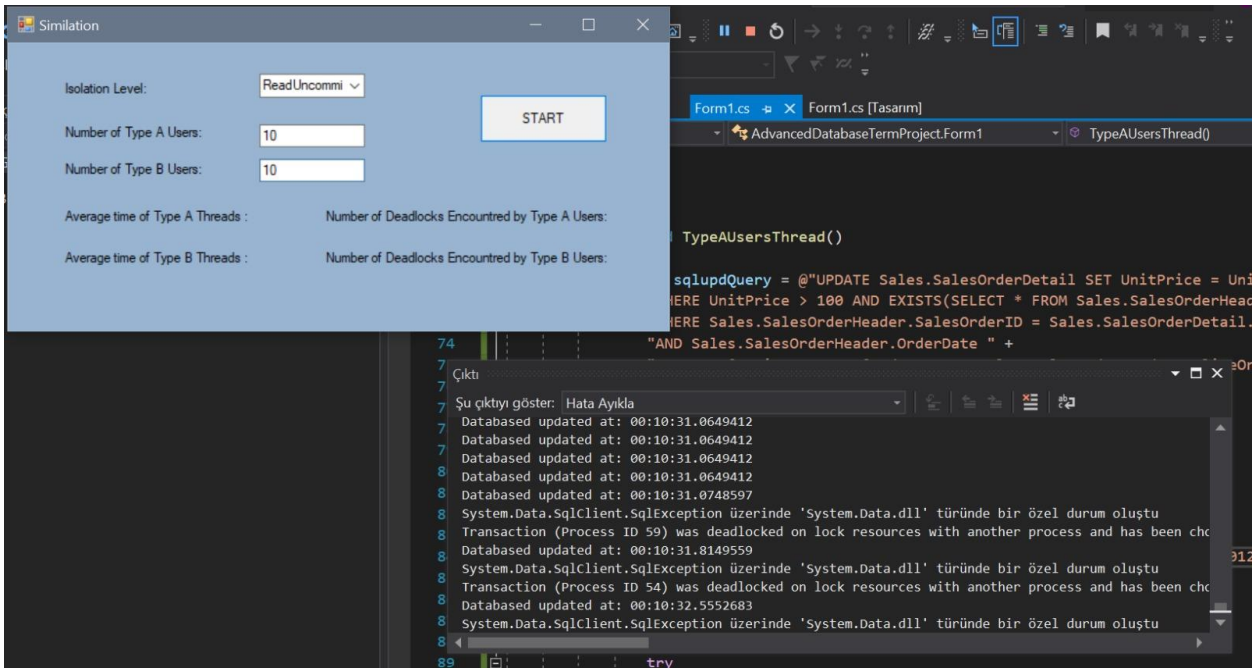
Isolation Level	**REPEATABLE READ**				
Number of Type A Users	Number of Type B Users	Average Time of Type A Threads (second)	Number of Deadlocks Encountered by Type A Users	Average Time of Type B Threads (second)	Number of Deadlocks Encountered by Type B Users
1	1	21,2	0	16,7	0
2	2	50,1	16	26,3	0
3	3	87,4	46	59,0	14
4	4	108,2	85	31,3	6
5	5	212,5	166	66,0	15
6	6	198,9	208	58,5	10
7	7	343,7	280	106,8	5
8	8	395,1	316	131,8	16
9	9	324,6	317	135,8	36
10	10	263,5	414	92,2	11
25	25	188,3	1695	30,5	0
40	40	515,2	2126	253,2	244
65	65	769,8	3379	373,7	0
80	80	1044,9	3927	753,1	514

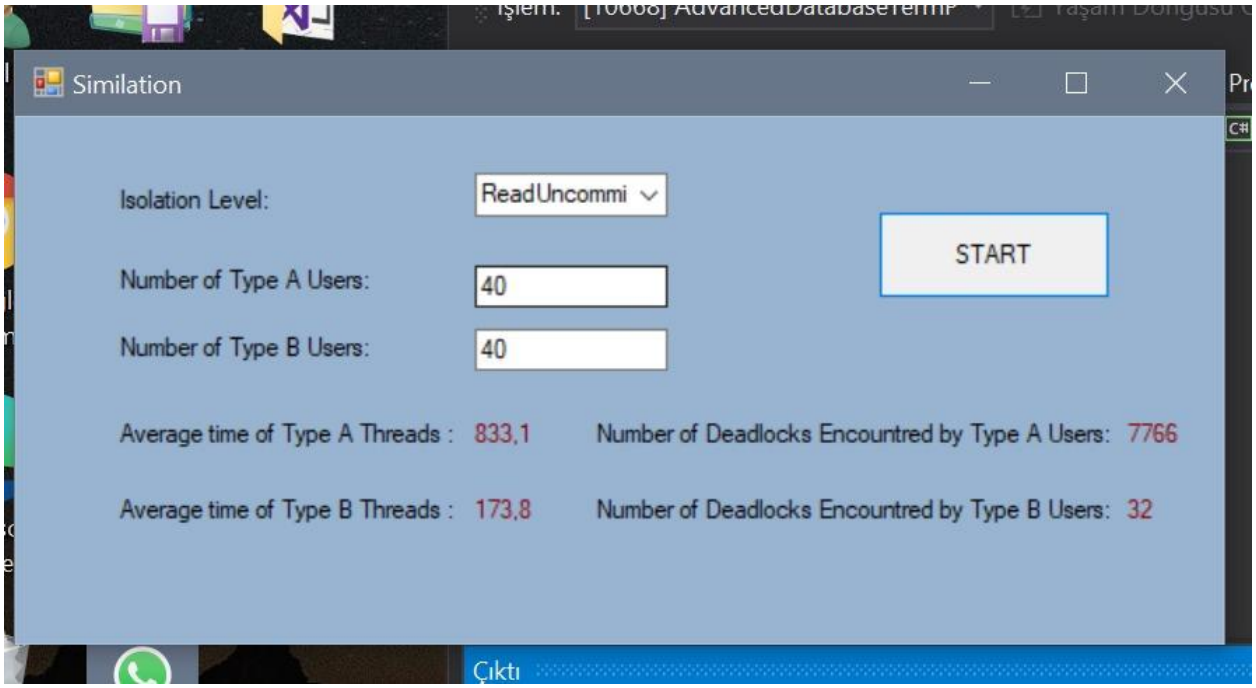
A different user who wants to update with the share lock he placed while the user is reading cannot update.That's why some lines have no deadlocks in user B, while others appear.This is because users b can also be deadlocked, as records that have not been committed cannot be accessed.

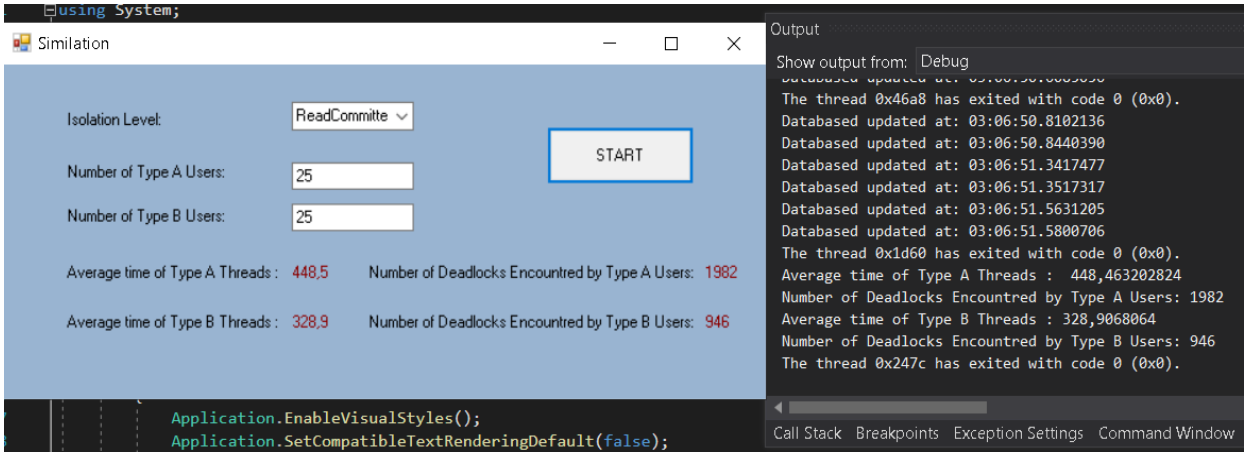
Isolation Level	**SERIALIZABLE**				
Number of Type A Users	Number of Type B Users	Average Time of Type A Threads (second)	Number of Deadlocks Encountered by Type A Users	Average Time of Type B Threads (second)	Number of Deadlocks Encountered by Type B Users
1	1	32,6	0	30,9	0
2	2	78,1	8	41,6	2
3	3	90,3	38	52,1	6
4	4	82,5	107	30,3	0
5	5	182,6	255	49,7	9
6	6	275,7	260	102,6	17
7	7	304,1	368	88,5	7
8	8	254,0	411	110,3	28
9	9	355,5	510	131,3	13
10	10	303,2	546	107,3	11
25	25	704,6	1440	342,5	88
40	40	735,9	1943	420,2	134
65	65	1117,4	2920	817,3	256
80	80	1326,2	2811	954,3	64

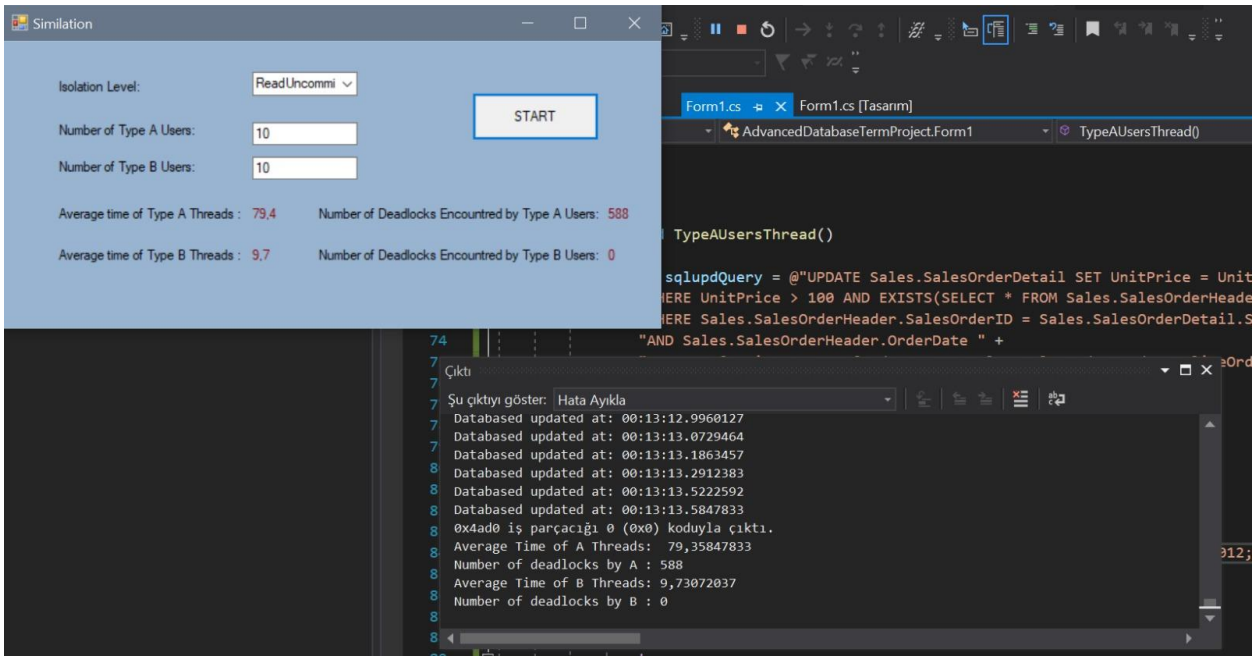
As the isolation level increases, we see that deadlock numbers increase for both threads, all users are waiting for some reason. At this isolation level, the number of deadlocks is greater, and the number of deadlocks increases as the number of users increases, so the average thread time naturally increases.

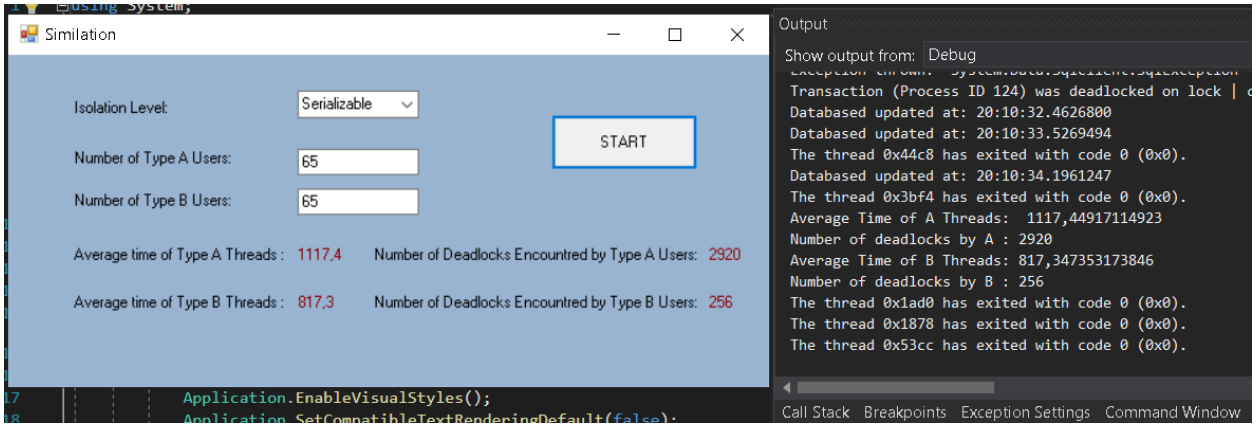
Screenshots From Our Simulation Program

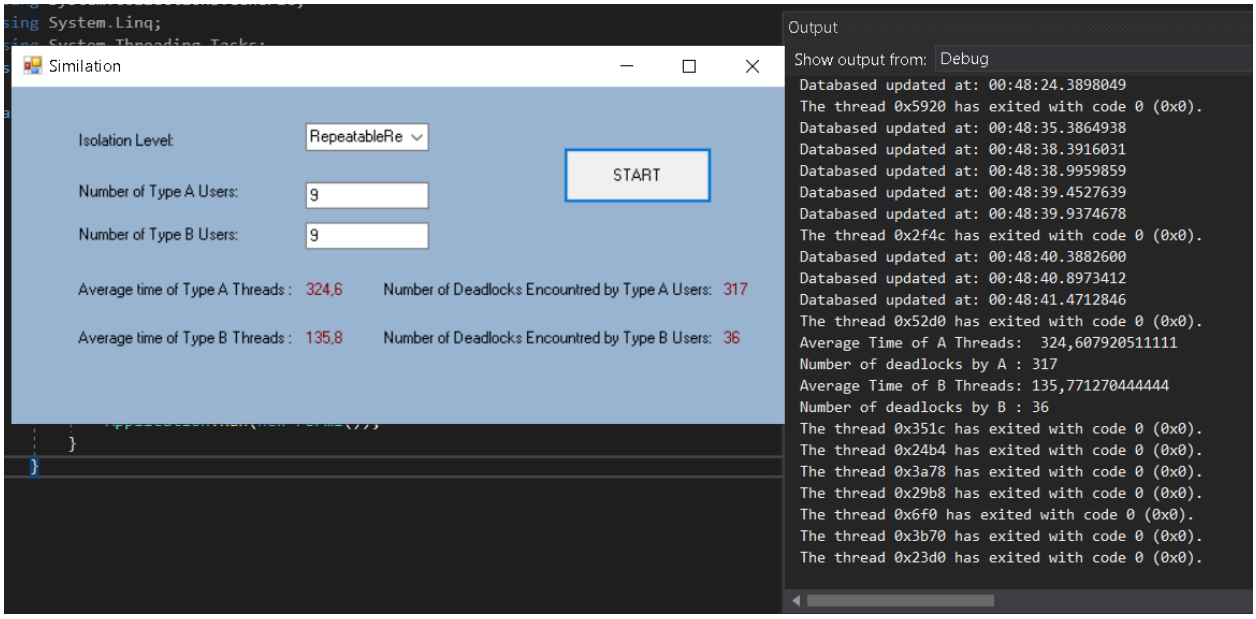


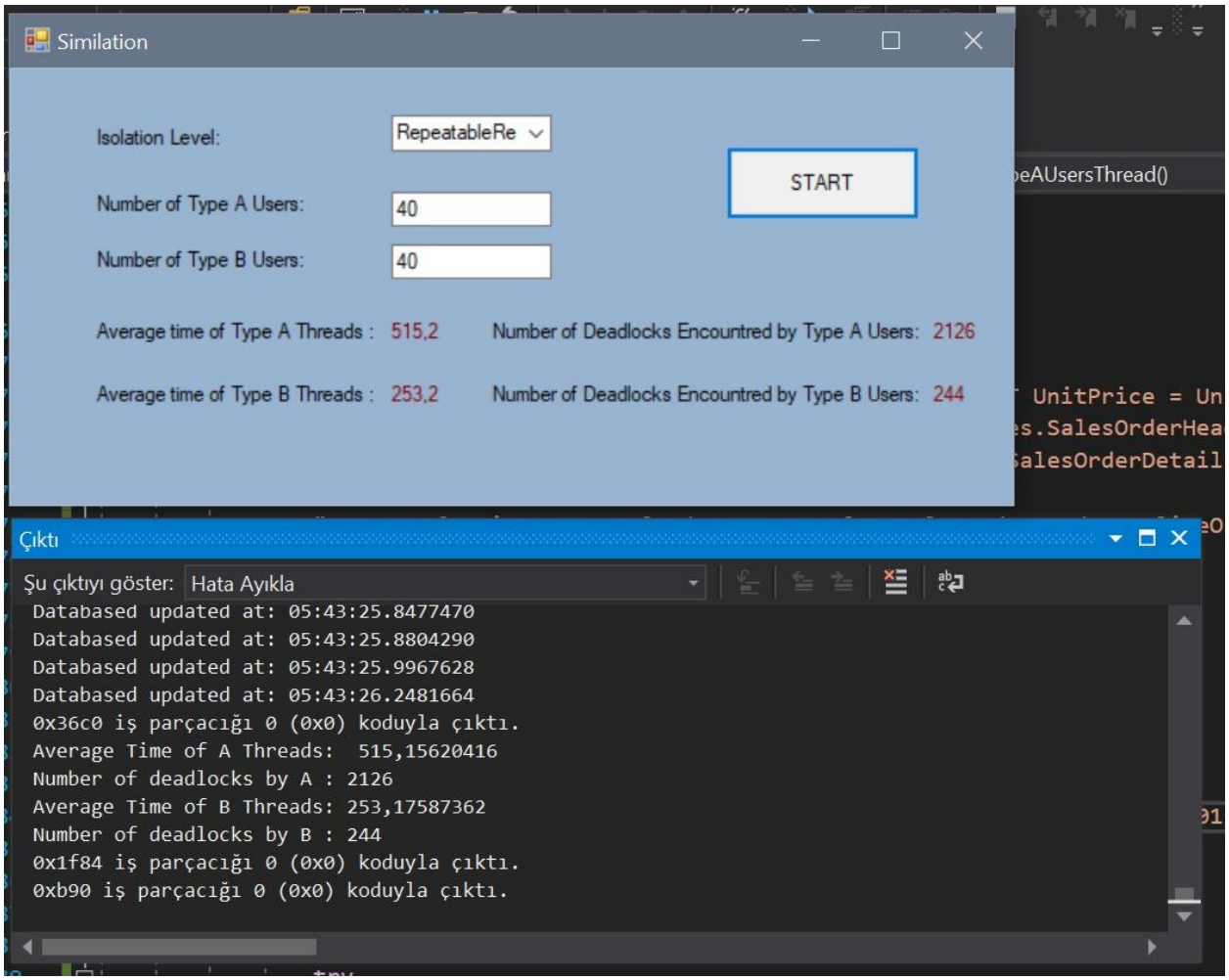


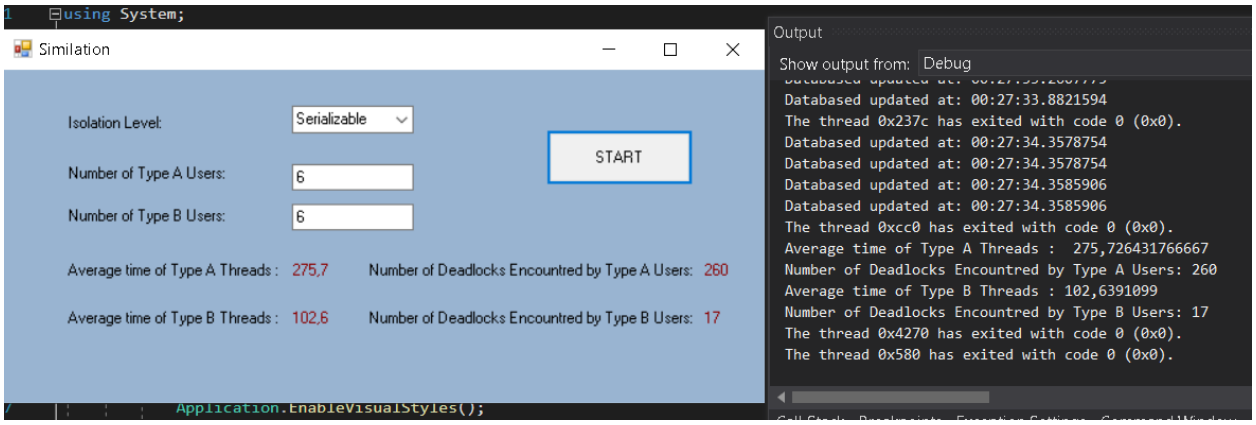

using System;
Similation
Isolation Level:
ReadCommitte
Number of Type A Users:
25
Number of Type B Users:
25
START
Average time of Type A Threads : 448,5
Number of Deadlocks Encountred by Type A Users: 1982
Average time of Type B Threads : 328,9
Number of Deadlocks Encountred by Type B Users: 946
Application.EnableVisualStyles();
Application.SetCompatibleTextRenderingDefault(false);
Output
Show output from: Debug
The thread 0x46a8 has exited with code 0 (0x0).
Databased updated at: 03:06:50.8102136
Databased updated at: 03:06:50.8440390
Databased updated at: 03:06:51.3417477
Databased updated at: 03:06:51.3517317
Databased updated at: 03:06:51.5631205
Databased updated at: 03:06:51.5800706
The thread 0x1d60 has exited with code 0 (0x0).
Average time of Type A Threads : 448,463202824
Number of Deadlocks Encountred by Type A Users: 1982
Average time of Type B Threads : 328,9068064
Number of Deadlocks Encountred by Type B Users: 946
The thread 0x247c has exited with code 0 (0x0).
Call Stack
Breakpoints
Exception Settings
Command Window

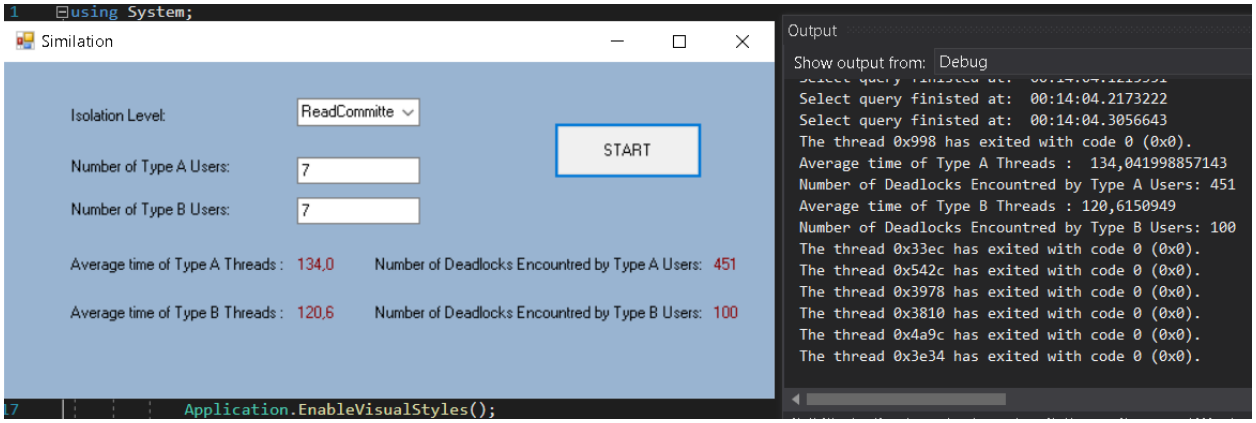

Similation
Isolation Level:
ReadUncommi
Number of Type A Users:
10
Number of Type B Users:
10
START
Average time of Type A Threads : 79,4
Number of Deadlocks Encountred by Type A Users: 588
Average time of Type B Threads : 9,7
Number of Deadlocks Encountred by Type B Users: 0
Form1.cs
Form1.cs [Tasarım]
AdvancedDatabaseTermProject.Form1
TypeAUsersThread()
TypeAUsersThread()
"AND Sales.SalesOrderHeader.OrderDate " +
Çıktı
Şu çıktıyı göster: Hata Ayıkla
Databased updated at: 00:13:12.9960127
Databased updated at: 00:13:13.0729464
Databased updated at: 00:13:13.1863457
Databased updated at: 00:13:13.2912383
Databased updated at: 00:13:13.5222592
Databased updated at: 00:13:13.5847833
0x4ad0 iş parçacığı 0 (0x0) koduyla çıktı.
Average Time of A Threads: 79,35847833
Number of deadlocks by A : 588
Average Time of B Threads: 9,73072037
Number of deadlocks by B : 0

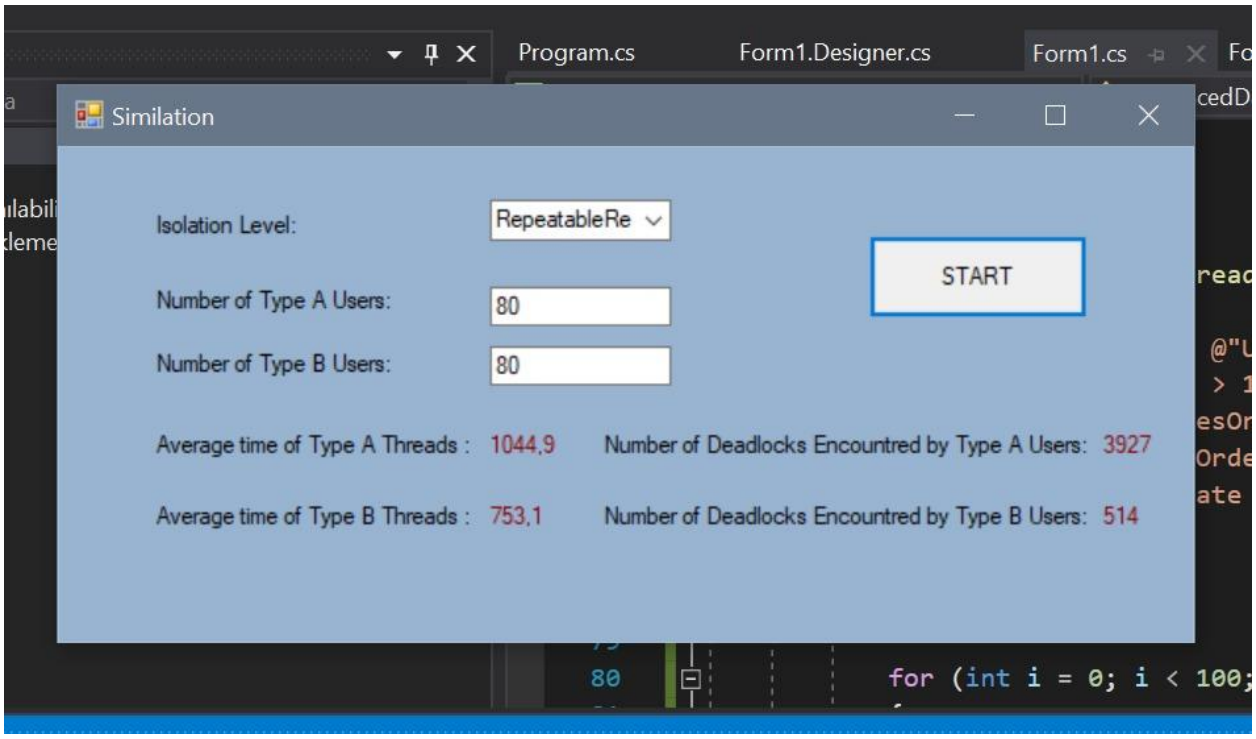

Similation
Isolation Level:
Serializable
Number of Type A Users:
65
Number of Type B Users:
65
START
Average time of Type A Threads : 1117,4
Number of Deadlocks Encountred by Type A Users: 2920
Average time of Type B Threads : 817,3
Number of Deadlocks Encountred by Type B Users: 256
Application.EnableVisualStyles();
Output
Show output from: Debug
Transaction (Process ID 124) was deadlocked on lock |
Databased updated at: 20:10:32.4626800
Databased updated at: 20:10:33.5269494
The thread 0x44c8 has exited with code 0 (0x0).
Databased updated at: 20:10:34.1961247
The thread 0x3bf4 has exited with code 0 (0x0).
Average Time of A Threads: 1117,44917114923
Number of deadlocks by A : 2920
Average Time of B Threads: 817,347353173846
Number of deadlocks by B : 256
The thread 0x1ad0 has exited with code 0 (0x0).
The thread 0x1878 has exited with code 0 (0x0).
The thread 0x53cc has exited with code 0 (0x0).
Call Stack
Breakpoints
Exception Settings
Command Window


Similation
Isolation Level:
RepeatableRe
Number of Type A Users:
9
Number of Type B Users:
9
START
Average time of Type A Threads : 324,6
Number of Deadlocks Encountred by Type A Users: 317
Average time of Type B Threads : 135,8
Number of Deadlocks Encountred by Type B Users: 36
Output
Show output from: Debug
Databased updated at: 00:48:24.3898049
The thread 0x5920 has exited with code 0 (0x0).
Databased updated at: 00:48:35.3864938
Databased updated at: 00:48:38.3916031
Databased updated at: 00:48:38.9959859
Databased updated at: 00:48:39.4527639
Databased updated at: 00:48:39.9374678
The thread 0x2f4c has exited with code 0 (0x0).
Databased updated at: 00:48:40.3882600
Databased updated at: 00:48:40.8973412
Databased updated at: 00:48:41.4712846
The thread 0x52d0 has exited with code 0 (0x0).
Average Time of A Threads: 324,607920511111
Number of deadlocks by A : 317
Average Time of B Threads: 135,771270444444
Number of deadlocks by B : 36
The thread 0x351c has exited with code 0 (0x0).
The thread 0x24b4 has exited with code 0 (0x0).
The thread 0x3a78 has exited with code 0 (0x0).
The thread 0x29b8 has exited with code 0 (0x0).
The thread 0x6f0 has exited with code 0 (0x0).
The thread 0x3b70 has exited with code 0 (0x0).
The thread 0x23d0 has exited with code 0 (0x0).


Similation
Isolation Level:
RepeatableRe
Number of Type A Users:
40
Number of Type B Users:
40
START
Average time of Type A Threads : 515,2
Number of Deadlocks Encountred by Type A Users: 2126
Average time of Type B Threads : 253,2
Number of Deadlocks Encountred by Type B Users: 244
Çıktı
Şu çıktıyı göster: Hata Ayıkla
Databased updated at: 05:43:25.8477470
Databased updated at: 05:43:25.8804290
Databased updated at: 05:43:25.9967628
Databased updated at: 05:43:26.2481664
0x36c0 iş parçacığı 0 (0x0) koduyla çıktı.
Average Time of A Threads: 515,15620416
Number of deadlocks by A : 2126
Average Time of B Threads: 253,17587362
Number of deadlocks by B : 244
0x1f84 iş parçacığı 0 (0x0) koduyla çıktı.
0xb90 iş parçacığı 0 (0x0) koduyla çıktı.


using System;
Similation
Isolation Level:
Serializable
START
Number of Type A Users:
6
Number of Type B Users:
6
Average time of Type A Threads : 275,7
Number of Deadlocks Encountred by Type A Users: 260
Average time of Type B Threads : 102,6
Number of Deadlocks Encountred by Type B Users: 17
Application.EnableVisualStyles();
Output
Show output from: Debug
Databased updated at: 00:27:33.8821594
The thread 0x237c has exited with code 0 (0x0).
Databased updated at: 00:27:34.3578754
Databased updated at: 00:27:34.3578754
Databased updated at: 00:27:34.3585906
Databased updated at: 00:27:34.3585906
The thread 0xcc0 has exited with code 0 (0x0).
Average time of Type A Threads : 275,726431766667
Number of Deadlocks Encountred by Type A Users: 260
Average time of Type B Threads : 102,6391099
Number of Deadlocks Encountred by Type B Users: 17
The thread 0x4270 has exited with code 0 (0x0).
The thread 0x580 has exited with code 0 (0x0).


using System;
Similation
Isolation Level:
ReadCommitte
START
Number of Type A Users:
7
Number of Type B Users:
7
Average time of Type A Threads : 134,0
Number of Deadlocks Encountred by Type A Users: 451
Average time of Type B Threads : 120,6
Number of Deadlocks Encountred by Type B Users: 100
Application.EnableVisualStyles();
Output
Show output from: Debug
Select query finisted at: 00:14:04.2173222
Select query finisted at: 00:14:04.3056643
The thread 0x998 has exited with code 0 (0x0).
Average time of Type A Threads : 134,041998857143
Number of Deadlocks Encountred by Type A Users: 451
Average time of Type B Threads : 120,6150949
Number of Deadlocks Encountred by Type B Users: 100
The thread 0x33ec has exited with code 0 (0x0).
The thread 0x542c has exited with code 0 (0x0).
The thread 0x3978 has exited with code 0 (0x0).
The thread 0x3810 has exited with code 0 (0x0).
The thread 0x4a9c has exited with code 0 (0x0).
The thread 0x3e34 has exited with code 0 (0x0).


Program.cs
Form1.Designer.cs
Form1.cs
Similation
Isolation Level:
RepeatableRe
START
Number of Type A Users:
80
Number of Type B Users:
80
Average time of Type A Threads : 1044,9
Number of Deadlocks Encountred by Type A Users: 3927
Average time of Type B Threads : 753,1
Number of Deadlocks Encountred by Type B Users: 514
for (int i = 0; i < 100;

The full source code of our simulation program

Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlClient;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Diagnostics;

namespace AdvancedDatabaseTermProject
{
   public partial class Form1 : Form
   {
      IsolationLevel ısolationLvl = new IsolationLevel();

      public Form1()
      {
         InitializeComponent();

      }
      private void isolationLvlDropBox_SelectedIndexChanged(object sender, EventArgs e)
      {

         if (isolationLvlComboBox.SelectedIndex == 0)
         {
            ısolationLvl = IsolationLevel.ReadUncommitted;
         }
         else if (isolationLvlComboBox.SelectedIndex == 1)
         {
            ısolationLvl = IsolationLevel.ReadCommitted;
         }
         else if (isolationLvlComboBox.SelectedIndex == 2)
         {
            ısolationLvl = IsolationLevel.RepeatableRead;
```

```
    }
    else if (isolationLvlComboBox.SelectedIndex == 3)
    {
        ısolationLvl = IsolationLevel.Serializable;
    }

}
public static SqlCommand sqlcommandUpd, sqlcommandSel;


int numberOfDeadlockbyTypeA, numberOfDeadlockbyTypeB = 0;
TimeSpan timeA;
TimeSpan timeB;
Stopwatch watch = new Stopwatch();


private void label1_Click(object sender, EventArgs e)
{

}

private void label1_Click_1(object sender, EventArgs e)
{

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

}


public void TypeAUsersThread()
{
    string sqlupdQuery = @"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 @BeginDate AND @EndDate AND Sales.SalesOrderHeader.OnlineOrderFlag = 1)";


    for (int i = 0; i < 100; i++)
    {
```

```
        DateTime beginTime = DateTime.Now;
        Random random = new Random();
                                                        string connectionString = "server
=DESKTOP-OEQJFLF;database=AdventureWorks2012;Trusted_Connection=True;";
        SqlConnection conn = new SqlConnection(connectionString);

        conn.Open();
        SqlTransaction transaction = conn.BeginTransaction(ısolationLvl);
        try
        {
          if (random.NextDouble() < 0.5)
          {
            SqlCommand sqlupdate = new SqlCommand(sqlupdQuery, conn, transaction);

            sqlupdate.Parameters.AddWithValue("@BeginDate", "20110101");
            sqlupdate.Parameters.AddWithValue("@EndDate", "20111231");
            sqlupdate.ExecuteNonQuery();
          }
          if (random.NextDouble() < 0.5)
          {
            SqlCommand sqlupdate = new SqlCommand(sqlupdQuery, conn, transaction);

            sqlupdate.Parameters.AddWithValue("@BeginDate", "20120101");
            sqlupdate.Parameters.AddWithValue("@EndDate", "20121231");
            sqlupdate.ExecuteNonQuery();
          }
          if (random.NextDouble() < 0.5)
          {
            SqlCommand sqlupdate = new SqlCommand(sqlupdQuery, conn, transaction);
            sqlupdate.Parameters.AddWithValue("@BeginDate", "20130101");
            sqlupdate.Parameters.AddWithValue("@EndDate", "20131231");
            sqlupdate.ExecuteNonQuery();
          }
          if (random.NextDouble() < 0.5)
          {
            SqlCommand sqlupdate = new SqlCommand(sqlupdQuery, conn, transaction);
            sqlupdate.Parameters.AddWithValue("@BeginDate", "20140101");
            sqlupdate.Parameters.AddWithValue("@EndDate", "20141231");
            sqlupdate.ExecuteNonQuery();

          }
          if (random.NextDouble() < 0.5)
```

```
            {
               SqlCommand sqlupdate = new SqlCommand(sqlupdQuery, conn, transaction);
               sqlupdate.Parameters.AddWithValue("@BeginDate", "20150101");
               sqlupdate.Parameters.AddWithValue("@EndDate", "20151231");
               sqlupdate.ExecuteNonQuery();
            }

            transaction.Commit();
         }

         catch (SqlException exc)
         {
            if (exc.Number == 1205)
            {
               numberOfDeadlockbyTypeA++;
               Console.WriteLine(exc.Message);
            }
            else
               Console.WriteLine(exc.Message);
         }
         DateTime endTime = DateTime.Now;
         timeA += endTime - beginTime;
         Console.WriteLine("Databased updated at: " + timeA);

         if (conn.State == ConnectionState.Open)
         {
            conn.Close();
         }

      }

   }

   public void TypeBUsersThread()
   {
      string sqlselQuery = @"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 @BeginDate AND @EndDate AND
Sales.SalesOrderHeader.OnlineOrderFlag = 1)";
        for (int i = 0; i < 100; i++)
        {
           DateTime beginTime = DateTime.Now;
           Random random = new Random();
                                                              string connectionString = "server
=DESKTOP-OEQJFLF;database=AdventureWorks2012;Trusted_Connection=True;";
           SqlConnection conn = new SqlConnection(connectionString);


           conn.Open();
           SqlTransaction transaction = conn.BeginTransaction(ısolationLvl);

           try
           {
              if (random.NextDouble() < 0.5)
              {
                 SqlCommand sqlselect = new SqlCommand(sqlselQuery, conn, transaction);
                 sqlselect.Parameters.AddWithValue("@BeginDate", "20110101");
                 sqlselect.Parameters.AddWithValue("@EndDate", "20111231");
                 sqlselect.ExecuteNonQuery();
              }
              if (random.NextDouble() < 0.5)
              {
                 SqlCommand sqlselect = new SqlCommand(sqlselQuery, conn, transaction);
                 sqlselect.Parameters.AddWithValue("@BeginDate", "20120101");
                 sqlselect.Parameters.AddWithValue("@EndDate", "20121231");
                 sqlselect.ExecuteNonQuery();
              }
              if (random.NextDouble() < 0.5)
              {
                 SqlCommand sqlselect = new SqlCommand(sqlselQuery, conn, transaction);
                 sqlselect.Parameters.AddWithValue("@BeginDate", "20130101");
                 sqlselect.Parameters.AddWithValue("@EndDate", "20131231");
                 sqlselect.ExecuteNonQuery();
              }
              if (random.NextDouble() < 0.5)
              {
                 SqlCommand sqlselect = new SqlCommand(sqlselQuery, conn, transaction);
                 sqlselect.Parameters.AddWithValue("@BeginDate", "20140101");
                 sqlselect.Parameters.AddWithValue("@EndDate", "20141231");
```

```
            sqlselect.ExecuteNonQuery();
        }
        if (random.NextDouble() < 0.5)
        {
            SqlCommand sqlselect = new SqlCommand(sqlselQuery, conn, transaction);
            sqlselect.Parameters.AddWithValue("@BeginDate", "20150101");
            sqlselect.Parameters.AddWithValue("@EndDate", "20151231");
            sqlselect.ExecuteNonQuery();
        }

        transaction.Commit();
    }

    catch (SqlException exc)
    {

        if (exc.Number == 1205)
        {
            numberOfDeadlockbyTypeB++;
            Console.WriteLine(exc.Message);
        }
        else
            Console.WriteLine(exc.Message);

    }
    DateTime endTime = DateTime.Now;
    timeB += endTime - beginTime;
    Console.WriteLine("Select query finisted at:  " + timeB);


    if (conn.State == ConnectionState.Open)
    {
        conn.Close();
    }

  }

}

private void typeAtextBox_TextChanged(object sender, EventArgs e)
{
```

```
}

private void button1_Click(object sender, EventArgs e)
{

   Random random = new Random();
   List<Thread> Threads = new List<Thread>();

   int numberofTypeAUsers = Int32.Parse(typeAtextBox.Text);
   int numberofTypeBUsers = Int32.Parse(typeBtextBox.Text);

   for (int i = 0; i < numberofTypeAUsers; i++)
   {
      Threads.Add(new Thread(new ThreadStart(TypeAUsersThread)));
   }
   for (int i = 0; i < numberofTypeBUsers; i++)
   {
      Threads.Add(new Thread(new ThreadStart(TypeBUsersThread)));
   }

   int n = Threads.Count;
   while (n > 1)
   {
      n--;
      int k = random.Next(n + 1);
      var value = Threads[k];
      Threads[k] = Threads[n];
      Threads[n] = value;
   }


   foreach (var thread in Threads)
   {
      thread.Start();
   }


   foreach (var thread in Threads)
   {
      thread.Join();
   }
```

```
                Console.WriteLine("Average time of Type A Threads :  " + timeA.TotalSeconds / numberofTypeAUsers);
                    Console.WriteLine("Number of Deadlocks Encountred by Type A Users: " + numberOfDeadlockbyTypeA);
        double avgtimeA = timeA.TotalSeconds / numberofTypeAUsers;

        label3.Text = avgtimeA.ToString("F1");
        label7.Text = numberOfDeadlockbyTypeA.ToString();

                  Console.WriteLine("Average time of Type B Threads : " + timeB.TotalSeconds / numberofTypeBUsers);
                    Console.WriteLine("Number of Deadlocks Encountred by Type B Users: " + numberOfDeadlockbyTypeB);
        double avgtimeB = timeB.TotalSeconds / numberofTypeBUsers;

        label4.Text = avgtimeB.ToString("F1");
        label8.Text = numberOfDeadlockbyTypeB.ToString();
    }

    private void textBoxA_TextChanged(object sender, EventArgs e)
    {

    }

    private void AvgTimeTypeAlabel_Click(object sender, EventArgs e)
    {

    }

  }
}
```

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace AdvancedDatabaseTermProject
{
   static class Program
   {
      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      [STAThread]
      static void Main()
      {
         Application.EnableVisualStyles();
         Application.SetCompatibleTextRenderingDefault(false);
         Application.Run(new Form1());
      }
   }
}
```