ADVANCED TOPICS IN DATABASE SYSTEMS

Database Optimization Simulation Program

Prepared by Harun Yahya ÜNAL

18/05/2025

This project is the second 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 with some optimization.

Three queries were used:

1/3 Query:
Select Order Date, Shipment Address State Name, Shipment Address City Name, Total Order Quantity, Total Order Line Total from Online orders between 1 Jan 2013 and 31 Dec 2013

```
SELECT  SOH.OrderDate,
        PROV.Name AS StateProvinceName,
        ADDR.City,
        SUM(SOD.OrderQty) AS TotalOrderQty,
        SUM(SOD.LineTotal) AS TotalLineTotal
  FROM Sales.SalesOrderDetail SOD
  INNER JOIN Sales.SalesOrderHeader SOH
     ON SOH.SalesOrderID = SOD.SalesOrderID
  INNER JOIN Person.Address ADDR
    ON ADDR.AddressID = SOH.ShipToAddressID
  INNER JOIN Person.StateProvince PROV
    ON PROV.StateProvinceID = ADDR.StateProvinceID
WHERE SOH.OrderDate BETWEEN '20130101' AND '20131231'
   AND SOH.OnlineOrderFlag = 1
GROUP BY SOH.OrderDate, PROV.Name, ADDR.City
 ORDER BY SOH.OrderDate, PROV.Name, ADDR.City
```

2/3 Query:
Select Order Date, Product Category Name, Total Order Quantity, Total Order Line Total from Online orders between 1 Jan 2013 and 31 Dec 2013 of the products with MakeFlag = 1 or FinishedGoodsFlag = 1, and color Black or Yellow.

```
SELECT  SOH.OrderDate,
        CAT.Name as CategoryName,
        SUM(SOD.OrderQty) AS TotalOrderQty,
        SUM(SOD.LineTotal) AS TotalLineTotal
  FROM Sales.SalesOrderDetail SOD
  INNER JOIN Sales.SalesOrderHeader SOH
     ON SOH.SalesOrderID = SOD.SalesOrderID
  INNER JOIN Production.Product P
     ON P.ProductID = SOD.ProductID
  INNER JOIN Production.ProductSubcategory SUBCAT
     ON SUBCAT.ProductCategoryID = P.ProductSubcategoryID
  INNER JOIN Production.ProductCategory CAT
     ON CAT.ProductCategoryID = SUBCAT.ProductSubcategoryID
WHERE SOH.OrderDate BETWEEN '20130101' AND '20131231'
```

```
    AND SOH.OnlineOrderFlag = 1
    AND (P.MakeFlag = 1 OR P.FinishedGoodsFlag = 1)
    AND P.Color IN ('Black', 'Yellow')
GROUP BY SOH.OrderDate, CAT.Name
 ORDER BY SOH.OrderDate, CAT.Name
```

3/3 Query:

Select Store Name, Product Category Name, Total Order Quantity, Total Order Line Total from orders (not online but from physical stores) between 1 Jan 2013 and 31 Dec 2013 of the products with MakeFlag = 1 or FinishedGoodsFlag = 1, and color Black or Yellow.

```
SELECT  STOR.Name as StoreName,
          CAT.Name as CategoryName,
          SUM(SOD.OrderQty) AS TotalOrderQty,
          SUM(SOD.LineTotal) AS TotalLineTotal
          FROM Sales.SalesOrderDetail SOD
  INNER JOIN Sales.SalesOrderHeader SOH
     ON SOH.SalesOrderID = SOD.SalesOrderID
  INNER JOIN Production.Product P
     ON P.ProductID = SOD.ProductID
  INNER JOIN Production.ProductSubcategory SUBCAT
     ON SUBCAT.ProductCategoryID = P.ProductSubcategoryID
  INNER JOIN Production.ProductCategory CAT
     ON CAT.ProductCategoryID = SUBCAT.ProductSubcategoryID
  INNER JOIN Sales.Customer CUST
     ON CUST.CustomerID = SOH.CustomerID
  INNER JOIN Sales.Store STOR
     ON STOR.BusinessEntityID = CUST.StoreID
WHERE SOH.OrderDate BETWEEN '20130101' AND '20131231'
    AND SOH.OnlineOrderFlag = 0
    AND (P.MakeFlag = 1 OR P.FinishedGoodsFlag = 1)
    AND P.Color IN ('Black', 'Yellow')
GROUP BY STOR.Name, CAT.Name
 ORDER BY STOR.Name, CAT.Name
```

The simulation program is built with multi-threading by using C# and Windows Forms. Each query structure runs at least 100 times. During the simulation, the program measures the time for each thread and the number of effected rows by the queries.

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

Query 1

- TOP 10 Records:

Order Date	State Province Name	City	Total Order Qty	Total Line Total
01.01.2013	California	Burbank	1	782,990000
01.01.2013	England	Oxon	1	2181,562500
01.01.2013	New South Wales	Malabar	1	1000,437500
01.01.2013	New South Wales	Silverwater	1	2049,098200
01.01.2013	New South Wales	Sydney	1	2181,562500
01.01.2013	Nordrhein-Westfalen	Paderborn	1	2443,350000
01.01.2013	Nordrhein-Westfalen	Solingen	1	2443,350000
01.01.2013	Oregon	Lebanon	1	2049,098200
01.01.2013	South Australia	Cloverdale	1	2049,098200
01.01.2013	Tasmania	Hobart	1	2443,350000

- Used Indexes for Optimization:

```
CREATE INDEX IDX_SOH_OrderDate_OnlineOrderFlag ON Sales.SalesOrderHeader(OrderDate, OnlineOrderFlag);

CREATE INDEX IDX_SOD_SalesOrderID ON Sales.SalesOrderDetail(SalesOrderID);

CREATE INDEX IDX_ADDR_AddressID_StateProvinceID ON Person.Address(AddressID, StateProvinceID);

CREATE INDEX IDX_PROV_StateProvinceID ON Person.StateProvince(StateProvinceID);
```

For the optimization of **Query 1**, I first created the index “IDX_SOH_OrderDate_OnlineOrderFlag” on the **OrderDate** and **OnlineOrderFlag** columns in the **SalesOrderHeader** table. These columns are heavily used for filtering online orders within a specific date range. By indexing them, SQL Server quickly finds the relevant rows without scanning the entire table.

I also created the index “IDX_SOD_SalesOrderID” on the **SalesOrderID** column in the **SalesOrderDetail** table. This index speeds up the join operation with **SalesOrderHeader**, as it allows direct access to the matching rows instead of a full table scan.

Additionally, I added the index “IDX_ADDR_AddressID_StateProvinceID” on the **AddressID** and **StateProvinceID** columns in the **Address** table. This optimization reduces the time required to join with **SalesOrderHeader** for retrieving shipment addresses, as the database can efficiently locate the rows.

Finally, I created the index “IDX_PROV_StateProvinceID” on the **StateProvinceID** column in the **StateProvince** table. This index accelerates the join with **Address**, allowing SQL Server to directly fetch the required state information during the query.

After applying these optimizations, I measured the execution time and surprisingly, the average query time increased from **5121 ms** to **5640 ms**. This result was unexpected, as the optimizations were intended to improve performance. I believe this may be due to the database engine choosing a suboptimal query plan or the additional overhead of maintaining the new indexes. To understand the situation better, by analyzing the **Query Execution Plan**, it can be checked whether the indexes are fully utilized.

- Before Optimization:

Total Time	Average Time	Number of Measurements	Number of Effected Rows
512122 ms	5121 ms	100	10899

- After Optimization:

Total Time	Average Time	Number of Measurements	Number of Effected Rows
564041 ms	5640 ms	100	10899

Note: I didn't add all 100 records to keep the report clean and tidy. The simulation program shows the duration time after each transaction. And also, after 100 transactions, it shows the total time and average time.

Query 2

- TOP 10 Records:

Order Date	Category Name	Total Order Qty	Total Line Total
01.01.2013	Accessories	4	6146,552500
01.01.2013	Bikes	3	6147,294600
01.01.2013	Clothing	3	6147,294600
01.01.2013	Components	3	6147,294600
02.01.2013	Accessories	5	4349,845000
02.01.2013	Bikes	2	4098,196400
02.01.2013	Clothing	2	4098,196400
02.01.2013	Components	2	4098,196400
03.01.2013	Accessories	6	10727,125000
03.01.2013	Bikes	1	2049,098200

- Used Indexes for Optimization:

```
CREATE INDEX IDX_SOH_OrderDate_OnlineOrderFlag_2 ON Sales.SalesOrderHeader(OrderDate, OnlineOrderFlag);

CREATE INDEX IDX_PRODUCT_Flags_Color ON Production.Product(ProductID, MakeFlag, FinishedGoodsFlag, Color);

CREATE INDEX IDX_SUBCAT_ProductCategoryID ON Production.ProductSubcategory(ProductSubcategoryID);

CREATE INDEX IDX_CAT_ProductSubcategoryID ON Production.ProductCategory(ProductCategoryID);
```

For the optimization of **Query 2**, I first created the index “IDX_SOH_OrderDate_OnlineOrderFlag_2” on the **OrderDate** and **OnlineOrderFlag** columns in the **SalesOrderHeader** table. These columns are heavily used to filter online orders within a specific date range. By indexing them, SQL Server can directly access the relevant rows without scanning the entire table, which reduces the search time significantly.

Next, I added the index “IDX_PRODUCT_Flags_Color” on the **ProductID**, **MakeFlag**, **FinishedGoodsFlag**, and **Color** columns in the **Production.Product** table. This index optimizes the filtering of products with specific flags and colors. Normally, SQL Server would need to scan the whole product list to find matching items, but with this index, it quickly identifies the required products, enhancing the overall speed of the query.

Additionally, I created the index “IDX_SUBCAT_ProductCategoryID” on the **ProductSubcategoryID** column in the **Production.ProductSubcategory** table. This column is used in join operations with **Product** to retrieve subcategory information. The index helps the database engine to find matching subcategories faster during the join process.

Finally, I added the index “IDX_CAT_ProductSubcategoryID” on the **ProductCategoryID** column in the **Production.ProductCategory** table. This index is crucial for optimizing the join with **ProductSubcategory** to fetch the category names. Without this index, SQL Server would perform a full scan of the category list, which would slow down the join operation.

After applying these optimizations, the average query time improved from **2608 ms** to **2327 ms**. This was an expected result, showing that the indexes were effective in reducing the query time. By creating

targeted indexes, SQL Server was able to access the required rows directly, minimizing the need for full table scans and speeding up join operations.

- Before Optimization:

Total Time	Average Time	Number of Measurements	Number of Effected Rows
260893 ms	2608 ms	100	1360

- After Optimization:

Total Time	Average Time	Number of Measurements	Number of Effected Rows
232788 ms	2327 ms	100	1360

Note: I didn't add all 100 records to keep the report clean and tidy. The simulation program shows the duration time after each transaction. And also, after 100 transactions, it shows the total time and average time.

Query 3

- TOP 10 Records:

Store Name	Category Name	Total Order Qty	Total Line Total
A Great Bicycle Company	Accessories	1	469,794000
Active Life Toys	Accessories	54	39726,255000
Activity Center	Accessories	18	15478,908000
Advanced Bike Components	Accessories	139	103831,976092
Affordable Sports Equipment	Accessories	34	24699,504000
Area Bike Accessories	Accessories	207	168791,830500
Area Sheet Metal Supply	Accessories	1	323,994000
Associated Bikes	Bikes	1	647,994000
Associated Bikes	Clothing	1	647,994000
Associated Bikes	Components	1	647,994000

- Used Indexes for Optimization:

```
CREATE INDEX IDX_SOH_OrderDate_OnlineOrderFlag_3 ON Sales.SalesOrderHeader(OrderDate, OnlineOrderFlag);

CREATE INDEX IDX_SOD_SalesOrderID_3 ON Sales.SalesOrderDetail(SalesOrderID);

CREATE INDEX IDX_CUST_StoreID ON Sales.Customer(CustomerID, StoreID);

CREATE INDEX IDX_STOR_BusinessEntityID ON Sales.Store(BusinessEntityID);
```

For the optimization of **Query 3**, I first created the index “IDX_SOH_OrderDate_OnlineOrderFlag_3” on the **OrderDate** and **OnlineOrderFlag** columns in the **SalesOrderHeader** table. These columns are crucial for filtering physical store orders within a specific date range. By indexing them, SQL Server can directly access the required rows, minimizing the need for a full table scan and speeding up the filtering process.

I also added the index “IDX_SOD_SalesOrderID_3” on the **SalesOrderID** column in the **SalesOrderDetail** table. This index optimizes the join operation with **SalesOrderHeader**, enabling SQL Server to quickly find the matching order details without scanning the entire table.

Additionally, I created the index “IDX_CUST_StoreID” on the **CustomerID** and **StoreID** columns in the **Sales.Customer** table. This optimization helps the database engine efficiently match customers with their related stores during the join operation.

Finally, the index “IDX_STOR_BusinessEntityID” was created on the **BusinessEntityID** column in the **Sales.Store** table. This index speeds up the join with **Customer** by directly accessing the required store information, avoiding a full table scan.

After applying these optimizations, the average query time slightly increased from **2847 ms** to **2859 ms**. This result was unexpected, as the optimizations were intended to improve performance. This may be due to the database choosing a suboptimal query plan or the overhead of maintaining the new indexes. A detailed analysis of the **Query Execution Plan** is required to understand if the indexes are fully utilized.

- Before Optimization:

Total Time	Average Time	Number of Measurements	Number of Effected Rows
284713 ms	2847 ms	100	656

- After Optimization:

Total Time	Average Time	Number of Measurements	Number of Effected Rows
285906 ms	2859 ms	100	656

Note: I didn't add all 100 records to keep the report clean and tidy. The simulation program shows the duration time after each transaction. And also, after 100 transactions, it shows the total time and average time.

Source Codes

DatabaseOperations.cs

```
using Advanced_Database_Topics_II.Models;
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Linq;
using System.Threading.Tasks;

namespace Advanced_Database_Topics_II
{
   public class DatabaseOperations
   {
      private const string QUERY1 = "SELECT SOH.OrderDate," +
         "                          PROV.Name AS StateProvinceName, " +
         "                          ADDR.City," +
         "                          SUM(SOD.OrderQty) AS TotalOrderQty," +
         "                          SUM(SOD.LineTotal) AS TotalLineTotal " +
         "                     FROM Sales.SalesOrderDetail SOD " +
         "                     INNER JOIN Sales.SalesOrderHeader SOH" +
         "                       ON SOH.SalesOrderID = SOD.SalesOrderID " +
         "                     INNER JOIN Person.Address ADDR " +
         "                       ON ADDR.AddressID = SOH.ShipToAddressID " +
         "                     INNER JOIN Person.StateProvince PROV " +
         "                       ON PROV.StateProvinceID = ADDR.StateProvinceID " +
         "                     WHERE SOH.OrderDate BETWEEN '20130101' AND '20131231' " +
         "                      AND SOH.OnlineOrderFlag = 1 " +
         "                     GROUP BY SOH.OrderDate, PROV.Name, ADDR.City " +
         "                     ORDER BY SOH.OrderDate, PROV.Name, ADDR.City";

      private const string QUERY2 = "SELECT SOH.OrderDate, " +
         "                          CAT.Name as CategoryName, " +
         "                          SUM(SOD.OrderQty) AS TotalOrderQty, " +
         "                          SUM(SOD.LineTotal) AS TotalLineTotal " +
         "                     FROM Sales.SalesOrderDetail SOD " +
         "                     INNER JOIN Sales.SalesOrderHeader SOH " +
         "                       ON SOH.SalesOrderID = SOD.SalesOrderID " +
         "                     INNER JOIN Production.Product P " +
         "                       ON P.ProductID = SOD.ProductID " +
         "                     INNER JOIN Production.ProductSubcategory SUBCAT " +
         "                       ON SUBCAT.ProductCategoryID = P.ProductSubcategoryID " +
         "                     INNER JOIN Production.ProductCategory CAT " +
```

```
    "                    ON CAT.ProductCategoryID = SUBCAT.ProductSubcategoryID " +
    "                  WHERE SOH.OrderDate BETWEEN '20130101' AND '20131231' " +
    "                    AND SOH.OnlineOrderFlag = 1 " +
    "                    AND (P.MakeFlag = 1 OR P.FinishedGoodsFlag = 1) " +
    "                    AND P.Color IN ('Black', 'Yellow') " +
    "                  GROUP BY SOH.OrderDate, CAT.Name " +
    "                  ORDER BY SOH.OrderDate, CAT.Name";

    private const string QUERY3 = "SELECT STOR.Name as StoreName, " +
    "                      CAT.Name as CategoryName, " +
    "                      SUM(SOD.OrderQty) AS TotalOrderQty, " +
    "                      SUM(SOD.LineTotal) AS TotalLineTotal " +
    "                   FROM Sales.SalesOrderDetail SOD " +
    "                  INNER JOIN Sales.SalesOrderHeader SOH " +
    "                    ON SOH.SalesOrderID = SOD.SalesOrderID " +
    "                  INNER JOIN Production.Product P " +
    "                    ON P.ProductID = SOD.ProductID " +
    "                  INNER JOIN Production.ProductSubcategory SUBCAT " +
    "                    ON SUBCAT.ProductCategoryID = P.ProductSubcategoryID " +
    "                  INNER JOIN Production.ProductCategory CAT " +
    "                    ON CAT.ProductCategoryID = SUBCAT.ProductSubcategoryID " +
    "                  INNER JOIN Sales.Customer CUST " +
    "                    ON CUST.CustomerID = SOH.CustomerID " +
    "                  INNER JOIN Sales.Store STOR " +
    "                    ON STOR.BusinessEntityID = CUST.StoreID " +
    "                  WHERE SOH.OrderDate BETWEEN '20130101' AND '20131231' " +
    "                    AND SOH.OnlineOrderFlag = 0 " +
    "                    AND (P.MakeFlag = 1 OR P.FinishedGoodsFlag = 1) " +
    "                    AND P.Color IN ('Black', 'Yellow') " +
    "                  GROUP BY STOR.Name, CAT.Name " +
    "                  ORDER BY STOR.Name, CAT.Name";

    private readonly string _connectionString = "Data Source=localhost\\SQLEXPRESS;Initial
Catalog=AdventureWorks2022;Integrated Security=True;MultipleActiveResultSets=True;Connect
Timeout=120;Max Pool Size=100000;";


    public async Task<(List<Query1Model>,int)> ExecuteQuery1Async()
    {
        var result = new List<Query1Model>();

        try
        {
            using (var connection = new SqlConnection(_connectionString))
            {
```

```
            await connection.OpenAsync();

            await ExequteDBCCAsync(connection);

            using (var command = new SqlCommand(QUERY1, connection))
            {
               using (var reader = await command.ExecuteReaderAsync())
               {
                  while (await reader.ReadAsync())
                  {
                     var orderDate = reader.GetDateTime(0);
                     var stateProvinceName = reader.GetString(1);
                     var city = reader.GetString(2);
                     var totalOrderQty = reader.GetInt32(3);
                     var totalLineTotal = reader.GetDecimal(4);

                     result.Add(new Query1Model
                     {
                        OrderDate = orderDate,
                        StateProvinceName = stateProvinceName,
                        City = city,
                        TotalOrderQty = totalOrderQty,
                        TotalLineTotal = totalLineTotal
                     });
                  }
               }
            }
         return (result.Take(10).ToList(), result.Count);
      }
      catch (Exception)
      {

         throw;
      }
   }

   public async Task<(List<Query2Model>,int)> ExecuteQuery2Async()
   {
      var result = new List<Query2Model>();

      try
      {
         using (var connection = new SqlConnection(_connectionString))
         {
```

```
            await connection.OpenAsync();

            await ExequteDBCCAsync(connection);

            using (var command = new SqlCommand(QUERY2, connection))
            {
                using (var reader = await command.ExecuteReaderAsync())
                {
                    while (await reader.ReadAsync())
                    {
                        var orderDate = reader.GetDateTime(0);
                        var categoryName = reader.GetString(1);
                        var totalOrderQty = reader.GetInt32(2);
                        var totalLineTotal = reader.GetDecimal(3);

                        result.Add(new Query2Model
                        {
                            OrderDate = orderDate,
                            CategoryName = categoryName,
                            TotalOrderQty = totalOrderQty,
                            TotalLineTotal = totalLineTotal
                        });
                    }
                }
            }
        }

        return (result.Take(10).ToList(), result.Count);
    }
    catch (Exception)
    {

        throw;
    }
}

public async Task<(List<Query3Model>,int)> ExecuteQuery3Async()
{
    var result = new List<Query3Model>();

    try
    {
        using (var connection = new SqlConnection(_connectionString))
        {
            await connection.OpenAsync();
```

```
        await ExequteDBCCAsync(connection);

        using (var command = new SqlCommand(QUERY3, connection))
        {
            using (var reader = await command.ExecuteReaderAsync())
            {
                while (await reader.ReadAsync())
                {
                    var storeName = reader.GetString(0);
                    var categoryName = reader.GetString(1);
                    var totalOrderQty = reader.GetInt32(2);
                    var totalLineTotal = reader.GetDecimal(3);

                    result.Add(new Query3Model
                    {
                        StoreName = storeName,
                        CategoryName = categoryName,
                        TotalOrderQty = totalOrderQty,
                        TotalLineTotal = totalLineTotal
                    });
                }
            }
        }

        return (result.Take(10).ToList(),result.Count);
    }
    catch (Exception)
    {

        throw;
    }
}

private async Task ExequteDBCCAsync(SqlConnection connection)
{
    string dbccQuery1 = "DBCC FREEPROCCACHE;";
    string dbccQuery2 = "DBCC DROPCLEANBUFFERS;";
    try
    {
        using (var command = new SqlCommand(dbccQuery1, connection))
        {
            await command.ExecuteNonQueryAsync();
        }
```

```
            using (var command = new SqlCommand(dbccQuery2, connection))
            {
                await command.ExecuteNonQueryAsync();
            }
        }
        catch (Exception)
        {

            throw;
        }
    }
  }
}
```

Simulation.cs

```
using Advanced_Database_Topics_II.Models;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Threading.Tasks;

namespace Advanced_Database_Topics_II
{
  public class Simulation
  {
    private readonly DatabaseOperations _databaseOperations;
    public Simulation()
    {
      _databaseOperations = new DatabaseOperations();
    }

    public async Task<(List<Query1Model>, int numOfReturnedRow, long duration)>
GenerateQuery1()
    {
      Stopwatch stopwatch = new Stopwatch();
      List<Task> queries = new List<Task>();

      stopwatch.Start();
      var query1Top10 = await _databaseOperations.ExecuteQuery1Async();
      int numOfReturnedRow = query1Top10.Item2;

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

```
            queries.Add(Task.Run(async () =>
            {
                try
                {
                    await _databaseOperations.ExecuteQuery1Async();
                }
                catch (Exception)
                {
                    throw;
                }
            }));
        }

        await Task.WhenAll(queries);
        stopwatch.Stop();
        var duration = stopwatch.ElapsedMilliseconds;
        return (query1Top10.Item1, numOfReturnedRow, duration);
    }

    public async Task<(List<Query2Model>, int numOfReturnedRow, long duration)>
GenerateQuery2()
    {
        List<Task> queries = new List<Task>();
        Stopwatch stopwatch = new Stopwatch();

        stopwatch.Start();
        var query2Top10 = await _databaseOperations.ExecuteQuery2Async();
        int numOfReturnedRow = query2Top10.Item2;

        for (int i = 0; i < 100; i++)
        {
            queries.Add(Task.Run(async () =>
            {
                try
                {
                    await _databaseOperations.ExecuteQuery2Async();
                }
                catch (Exception)
                {
                    throw;
                }
            }));
        }

        await Task.WhenAll(queries);
```

```
        stopwatch.Stop();
        var duration = stopwatch.ElapsedMilliseconds;
        return (query2Top10.Item1, numOfReturnedRow, duration);
    }

    public async Task<(List<Query3Model>, int numOfReturnedRow, long duration)>
GenerateQuery3()
    {
        List<Task> queries = new List<Task>();
        Stopwatch stopwatch = new Stopwatch();

        stopwatch.Start();
        var query3Top10 = await _databaseOperations.ExecuteQuery3Async();
        int numOfReturnedRow = query3Top10.Item2;

        for (int i = 0; i < 100; i++)
        {
            queries.Add(Task.Run(async () =>
            {
                try
                {
                    await _databaseOperations.ExecuteQuery3Async();
                }
                catch (Exception)
                {
                    throw;
                }
            }));
        }

        await Task.WhenAll(queries);
        stopwatch.Stop();
        var duration = stopwatch.ElapsedMilliseconds;
        return (query3Top10.Item1, numOfReturnedRow, duration);
    }
  }
}
```

SimulationPage.cs

```
using System;
using System.Diagnostics;
using System.Windows.Forms;
```

```
namespace Advanced_Database_Topics_II
{
   public partial class SimulationPage : Form
   {
      private readonly Simulation _simulation;
      public SimulationPage()
      {
         _simulation = new Simulation();
         InitializeComponent();
      }

      private async void btnSimulateQ1_Click(object sender, EventArgs e)
      {
         Stopwatch stopwatch = new Stopwatch();

         stopwatch.Start();
         for (int i = 0; i < 100; i++)
         {
            var query1Top10 = await _simulation.GenerateQuery1();
            dgwTop10.DataSource = query1Top10.Item1;
            lblEffectedRow.Text = query1Top10.Item2.ToString();
            lblDuration.Text = $"{query1Top10.Item3} ms";
            Console.WriteLine($"Query 1 - {i + 1} executed");
         }
         stopwatch.Stop();
         var duration = stopwatch.ElapsedMilliseconds;
         lblDuration.Text = $"Total: {duration} ms\nAverage: {duration/100} ms";
         Console.WriteLine("**********Simulation Over********");
      }

      private async void btnSimulateQ2_Click(object sender, EventArgs e)
      {
         Stopwatch stopwatch = new Stopwatch();

         stopwatch.Start();
         for (int i = 0; i < 100; i++)
         {
            var query2Top10 = await _simulation.GenerateQuery2();
            dgwTop10.DataSource = query2Top10.Item1;
            lblEffectedRow.Text = query2Top10.Item2.ToString();
            lblDuration.Text = $"{query2Top10.Item3} ms";
            Console.WriteLine($"Query 2 - {i + 1} executed");
         }
         stopwatch.Stop();
         var duration = stopwatch.ElapsedMilliseconds;
```

```
        lblDuration.Text = $"Total: {duration} ms\nAverage: {duration / 100} ms";
        Console.WriteLine("**********Simulation Over********");
    }

    private async void btnSimulateQ3_Click(object sender, EventArgs e)
    {
        Stopwatch stopwatch = new Stopwatch();

        stopwatch.Start();
        for (int i = 0; i < 100; i++)
        {
            var query3Top10 = await _simulation.GenerateQuery3();
            dgwTop10.DataSource = query3Top10.Item1;
            lblEffectedRow.Text = query3Top10.Item2.ToString();
            lblDuration.Text = $"{query3Top10.Item3} ms";
            Console.WriteLine($"Query 3 - {i + 1} executed");
        }
        stopwatch.Stop();
        var duration = stopwatch.ElapsedMilliseconds;
        lblDuration.Text = $"Total: {duration} ms\nAverage: {duration / 100} ms";
        Console.WriteLine("**********Simulation Over********");
    }
  }
}
```

Models:

```
using System;

namespace Advanced_Database_Topics_II.Models
{
  public class Query1Model
  {
    public DateTime OrderDate { get; set; }
    public string StateProvinceName { get; set; }
    public string City { get; set; }
    public int TotalOrderQty { get; set; }
    public decimal TotalLineTotal { get; set; }
  }

  public class Query2Model
  {
    public DateTime OrderDate { get; set; }
```

```
        public string CategoryName { get; set; }
        public int TotalOrderQty { get; set; }
        public decimal TotalLineTotal { get; set; }
    }

    public class Query3Model
    {
        public string StoreName { get; set; }
        public string CategoryName { get; set; }
        public int TotalOrderQty { get; set; }
        public decimal TotalLineTotal { get; set; }
    }
}
```