Лабораторная работа №9
Исследование способов оптимизации программного кода

1 Цель работы

1.1 Изучить методы оптимизации программного кода.

2 Литература

2.1 Фленов, М. Е. Библия C# / М. Е. Фленов. – 5-е изд. – Санкт-Петербург : БХВ-Петербург, 2022. – 464 с. – URL: https://ibooks.ru/bookshelf/380047/reading. – Режим доступа: для зарегистрир. пользователей. – Текст : электронный. – гл.12,14.

3 Подготовка к работе

3.1 Повторить теоретический материал (см.п.2).

3.2 Изучить описание лабораторной работы.

4 Основное оборудование

4.1 Персональный компьютер.

5 Задание

5.1 Оптимизация чтения из файла

Выполнить оптимизацию, используя асинхронное чтение из файла и увеличив размер буфера.

```
// Чтение из файла
string filePath = "data.txt";
project.ReadFromFile(filePath);

// Нерациональный метод для чтения из файла
public void ReadFromFile(string filePath)
{
	if (!File.Exists(filePath))
	{
		Console.WriteLine("File not found.");
		return;
	}

	// Оптимизация: добавить в конструктор параметр bufferSize: 8192
	// для увеличения размера буфера
	using (var reader = new StreamReader(filePath))
	{
		string line;
		while ((line = reader.ReadLine()) != null)
		{
			// Оптимизация: использовать асинхронные операции
			Console.WriteLine(line);
		}
	}
}
```

5.2 Оптимизация записи в файл

Выполнить оптимизацию, используя асинхронную запись в файл и увеличив размер буфера.

```
// Запись в файл
project.WriteToFile(filePath, "Some data to be written to the file");
```

```
// Нерациональный метод для записи в файл
public void WriteToFile(string filePath, string data)
{
	// Оптимизация: добавить в конструктор параметры Encoding.UTF8 и bufferSize: 8192
	// для увеличения размера буфера
	using (var writer = new StreamWriter(filePath, true))
	{
		// Оптимизация: заменить посимвольную запись записью всей строки в файл
		foreach (char c in data)
		{
			// Оптимизация: использовать асинхронные операции
			writer.Write(c);
		}
	}
}
```

5.3 Оптимизация вычислений

Выполнить оптимизацию, используя кэширование уже вычисленных значений.

Подсказка: для кэширования создать поле Dictionary<int, int> _fibonacciCache (ключ – число, значение – вычисленное значение для ключа). В методе проверять, есть ли ключ в словаре. Если есть – вернуть его, если нет – вычислить и добавить в словарь.

```
// Вычисление чисел Фибоначчи (сумма двух предыдущих значений: 0,1,1,2,3,5,8,...)
Console.WriteLine("Fibonacci Sequence:");
for (int i = 0; i < 20; i++)
{
	// Оптимизация: избежать повторного вычисления
	Console.WriteLine($"Fib({i}) = {project.Fibonacci(i)}");
}

// Нерациональный алгоритм для вычисления чисел Фибоначчи
public int Fibonacci(int n)
{
	if (n <= 1)
		return n;
	// Оптимизация: кэширование и улучшение алгоритма
	return Fibonacci(n - 1) + Fibonacci(n - 2);
}
```

Задания 5.4-5.8: Оптимизация работы с БД

Проект UserOptimizationExample представляет собой приложение, работающее с пользователями (User). Оно подключается к БД и позволяет:

- получать информацию обо всех пользователях, которые активны (IsActive).
- выполнять массовое добавление информации о пользователях.
- получать информацию о пользователях с их заказами.

Основные проблемы для оптимизации:

- избегание лишних запросов в цикле,
- использование проекции вместо загрузки полных объектов,
- оптимизация загрузки связанных данных (Include),
- отключение отслеживания для операций "только чтение" с AsNoTracking,
- группировка операций вставки пользователей с использованием AddRange,
- оптимизация работы с транзакциями,
- кэширование часто используемых данных.

5.4 Оптимизация метода GetActiveUsersAsync

Добавить AsNoTracking, чтобы избежать лишнего отслеживания сущностей, так как данные только читаются.

5.5 Оптимизация метода GetUsersWithOrdersAsync

Если нужны только имена пользователей и общее количество заказов, нужно использовать проекцию (Select), чтобы выбирать только необходимые данные, а не загружать весь объект.

5.6 Оптимизация метода AddUsersAsync

Вместо цикла, в котором вызывается SaveChangesAsync для каждого пользователя, использовать AddRange и один вызов SaveChangesAsync.

5.7 Оптимизация метода AddUsersAsync

Заключить всю работу по записи данных в БД в транзакцию, чтобы повысить атомарность операций:

using (var transaction = await context.Database.BeginTransactionAsync())

// работа с БД

...

await transaction.CommitAsync();

Проверить, что будет при передаче:

- корректных данных,
- некорректных данных,
- вначале корректных, затем некорректных данных.

5.8 Кэширование данных

Добавить в UserService:

- метод для кэширования Task<List<User>> GetCachedActiveUsersAsync
- поле для хранения продолжительности кэширования:

private readonly TimeSpan _cacheDuration = TimeSpan.FromMinutes(5);

В метод GetCachedActiveUsersAsync добавить кэширование для списка активных пользователей, чтобы сократить количество обращений к БД:

```
if (!_cache.TryGetValue("Название кэша", out кэшируемыеДанные))
{
      // заполнение кэшируемых данных
      ...
      _cache.Set("Название кэша", кэшируемыеДанные, длительность);
}
```

Созданный метод протестировать.

6 Порядок выполнения работы

6.1 Выполнить все задания из п.5 в одном решении.

При разработке считать, что пользователь ввел данные требуемого типа, остальные возможные ошибки обрабатывать.

При выполнении заданий использовать минимально возможное количество команд и переменных и выполнять форматирование и рефакторинг кода.

6.2 Ответить на контрольные вопросы.

7 Содержание отчета

7.1Титульный лист

7.2 Цель работы

7.3 Ответы на контрольные вопросы

7.4 Вывод

8 Контрольные вопросы

8.1 Что такое «оптимизация программного кода»?

8.2 Какова цель оптимизации программного кода?

8.3 Какие методы оптимизации программного кода применяются?

8.4 Почему асинхронный вызов относится к оптимизации?