Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

|  |
| --- |
| Институт информационных технологий и анализа данных |

наименование института

|  |
| --- |
| Допускаю к защите  Руководитель:  М.Д. Каташевцев |
| И.О. Фамилия |

Управление данными в высоконагруженных системах

наименование темы

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе по дисциплине

|  |
| --- |
| Управление данными |

|  |
| --- |
| 1.001.00.00 ПЗ |
| обозначение документа |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Выполнил студент группы | ИСМб 19-1 |  |  |  | Иванов И.И. |
|  | шифр группы |  | подпись |  | Фамилия И.О |
| Нормоконтроль |  |  |  |  | Каташевцев М.Д. |
|  |  |  | подпись |  | Фамилия И.О |

|  |  |
| --- | --- |
| Курсовая работа защищена с оценкой |  |

Иркутск 2021 г.

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**ИРКУТСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**ЗАДАНИЕ**

**НА КУРСОВУЮ РАБОТУ**

|  |  |  |
| --- | --- | --- |
| По курсу | Управление данными | |
| Студенту | Иванову Ивану Ивановичу | |
|  | (фамилия, инициалы) | |
| Тема работы: | Управление данными в высоконагруженных системах | | |
| Исходные данные: | | Вариант 1 | |
| Рекомендуемая литература: | | | |
| 1. Грубер М. Понимание SQL / Мартин Грубер. - Москва, 1993. – 291 c.  2. Петкович Д. Microsoft® SQL Server™ 2012. Руководство для начинающих: Пер с англ. / Душан Петкович. – СПБ.: БХВ-Петербург, 2013. – 816 с.  3. Управление Данными [Электронный ресурс] // kaboom.tealeaf.su : чаинка, 2021. URL: http://kaboom.tealeaf.su (дата обращения: 01.12.2021). | | | |
|  | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Графическая часть на | - | | листах. | |
| Дата выдачи задания |  | 08 / 11 / 2021 г. | | |
| Задание получил | | | |  | |  | Иванов И.И. |
|  | | | | подпись | |  | Фамилия И.О. |

|  |  |
| --- | --- |
| Дата представления работы руководителю | 21 / 12 / 2021 г. |
| Руководитель курсовой работы |  |  | Каташевцев М.Д. |
|  | подпись | Фамилия И.О. |

Содержание

[Введение 4](#_Toc89302436)

[1 Индивидуальный вариант 5](#_Toc89302437)

[2 Схема данных 6](#_Toc89302438)

[3 Внешний вид главного окна 7](#_Toc89302439)

[4 Генератор кошек 8](#_Toc89302440)

[4.1 Описание 8](#_Toc89302441)

[4.2 Код функции генерирующей кошек 8](#_Toc89302442)

[4.3 Код реакции на нажатие кнопки сгенерировать кошек 9](#_Toc89302443)

[5 Код генерации локаций 11](#_Toc89302444)

[5.1 Описание 11](#_Toc89302445)

[5.1 Код функции генерирующей здание 11](#_Toc89302446)

[5.3 Код функции генерирующей локации 13](#_Toc89302447)

[5.4 Код реакции на нажатие кнопки сгенерировать локации 13](#_Toc89302448)

[6 Статистика 15](#_Toc89302449)

[6.1 Описание 15](#_Toc89302450)

[6.2 Код запроса статистики 15](#_Toc89302451)

[6.3 Код тика таймера 16](#_Toc89302452)

[7 Описание работы интерфейса 17](#_Toc89302453)

[Заключение 21](#_Toc89302454)

[Список использованных источников 22](#_Toc89302455)

# Введение

В данной работе отрабатывается взаимодействие с базами данных в условиях высоконагруженных проектов с большим количество данных и конкурентных запросов.

В качестве СУБД используется Microsoft SQL Server 2019 Community. Microsoft SQL Server 2019 – система управления реляционными базами данных разработанная корпорацией Microsoft.

В условиях разработки крупных коммерческих приложений возникает необходимость обрабатывать огромные объемы данных. Размеры БД могут достигать десятков терабайт. Количество записей в одной таблице может уходить за миллиарды строк. К сожалению, трудно найти в открытых источниках ресурсы предоставляющие такой объем данных для личных целей. Однако, чтобы овладеть навыком оперирования большими данными надо с этими данными столкнуться.

В рамках данной курсовой работы разработано приложение на языке C#, которое позволяет сымитировать ситуацию работы с БД в большим количеством данных. C# – высокоуровневый язык программирования общего назначения, широко применяющийся при разработке энтерпрайз решений и имеющий тесную интеграцию с Microsoft SQL Server.

В данной работе разработан генератор данных, который по заданным параметрам позволяет автоматически заполнить таблицы в базе данных, а также нагенерировать данные и связи между строками таблиц.

Для имитации взаимодействия множества пользователей с приложением разработан механизм кокурентного выполнения запросов к БД. Таким образом разработанная система позволяет сымитировать параллельное выполнение сложных запросов.

Одним из важнейших принципов организации данных в БД является консистентность и целостность данных. Не смотря на увеличение пропускной способности современных каналов связи, а также повышения надежности современных серверов, все еще случаются перебои в работе оборудования, вызванные как природным, так и антропогенном влиянием на отлаженные системы. В случае передаче набора данных образующих единую логическую единицу, важно избежать моментов частичной передачи данных. Для решения этой проблемы современные СУБД предлагают механизм транзакций. Транзакция — это сущность которая может быть выполнена либо целиком и успешно, соблюдая целостность данных и независимо от параллельно идущих других транзакций, либо не выполнена вообще.

Microsoft SQL Server имеет встроенный механизм транзакций, язык C# позволяет вызывать запросы к БД оборачивая их в транзакции, тем самым добиваясь целостности данных. Разработанное в данной работе приложение имеет соответствующий флаг, включая и отключая который можно указать выполнять ли приложению запросы к базе в транзакции или нет.

# 1 Индивидуальный вариант

Реализовать генератор данных для БД «Кошачьи гонки».

База данных должна удовлетворять следующим требованиям

1. У всех таблиц должен быть суррогатный первичный ключ
2. Должна быть хотя бы одна таблица со связью (one-to-many)
3. Должна быть хотя бы одна таблица которая связывает две (связь many-to-many)
4. В таблице many-to-many помимо полей которые связывают две таблицы должно быть еще поле которое характеризует эту связь
5. Должна быть хотя бы одна таблица со связью на саму себя

Приложение должно удовлетворять следующим требованиям:

1. В программе должна быть возможность автоматически заполнить таблицу основной модели
2. В программе должна быть возможность автоматически заполнить иерархическую таблицу
3. В программе должна выводиться статистика по данным в таблицах
4. В программе должна быть возможность запустить генератор в режиме с транзакцией и без

# 2 Схема данных



# 3 Внешний вид главного окна

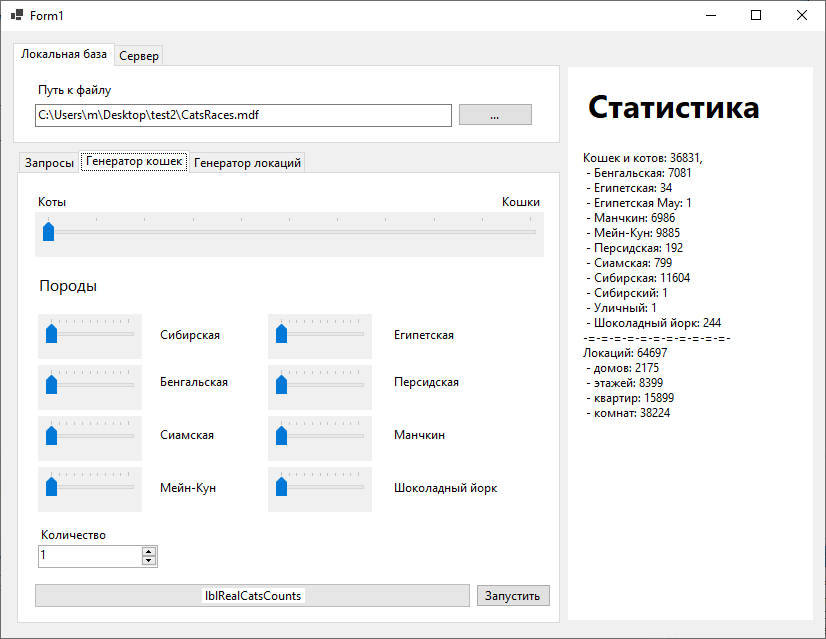


Рисунок 3.1

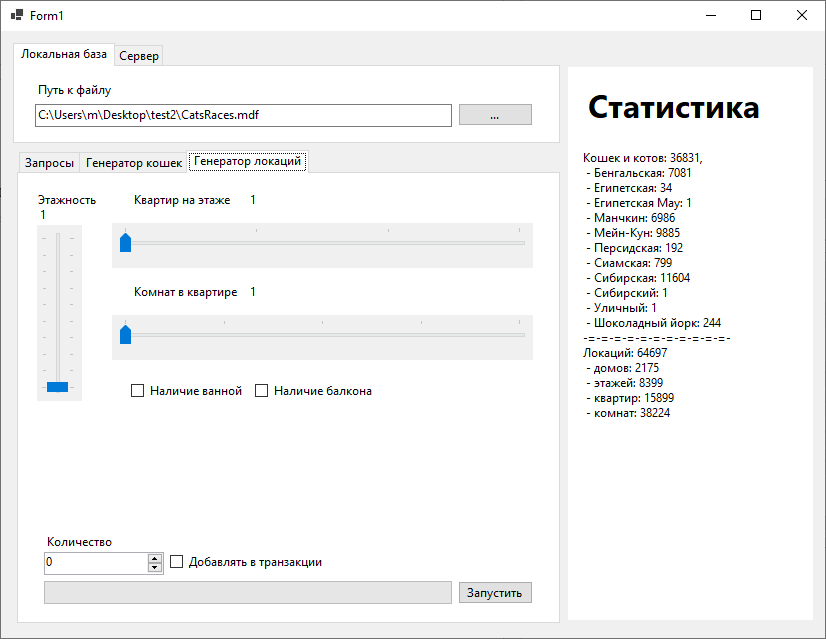


Рисунок 3.2

# 4 Генератор кошек

## 4.1 Описание

Генератор кошек позволяет указать количество экземпляров кошек, которые будут добавлены в базу данных. Есть возможность отрегулировать долю женских и мужских особей в общем распределении генерируемых данных. В дополнение к выше перечислимому можно настроить долю пород. Всего можно управлять восемью породами: Сибирская, Бенгальская, Сиамская, Мейн-кун, Египетская, Персидская, Манчкин, Шоколадный йорк

## 4.2 Код функции генерирующей кошек

CancellationTokenSource catGeneratorCanceletionToken = null;

private void GenerateCats(

SqlConnection connection,

Dictionary<string, Range> breedsProbablity,

int count,

int maleFemale,

IProgress<ProgressInfo> progress, // вот тут

CancellationTokenSource cancellationToken

)

{

using (connection) // тут теперь просто передаю переменную в using

{

connection.Open();

var transaction = connection.BeginTransaction();

Random rnd = new();

for (var i = 0; i < count; ++i)

{

string breed = null;

var k = rnd.NextDouble();

foreach (var item in breedsProbablity)

{

if (item.Value.CheckContains(k))

{

breed = item.Key;

break;

}

}

if (breed == null)

continue;

var isMale = ((double)maleFemale / 100) < rnd.NextDouble();

var gender = isMale ? "муж" : "жен";

var name = GenerateCatName(gender);

SqlCommand command = new($@"

INSERT INTO Cats(name, breed)

VALUES('{name}', '{breed}')"

, connection);

command.Transaction = transaction;

command.ExecuteNonQuery();

//progress?.Report(i + 1);

command = new("SELECT count(\*) FROM Cats", connection);

command.Transaction = transaction;

var info = $"Кошек в транзакции {command.ExecuteScalar()}";

progress?.Report(new ProgressInfo

{

value = i + 1,

info = info,

});

// если пришел запрос на завершение

if (cancellationToken.IsCancellationRequested)

{

return; // то выходим их функции

}

Thread.Sleep(30);

}

transaction.Commit();

}

}

## 4.3 Код реакции на нажатие кнопки сгенерировать кошек

private async void btnGenerateCats\_Click(object sender, EventArgs e)

{

// когда токен не равен нулю значит идет генерацию

if (catGeneratorCanceletionToken != null)

{

// отправляем запрос на отмену генерации

catGeneratorCanceletionToken.Cancel();

// выходим из функции

return;

}

// если токен равен нулю значит генерация не идет, создаем новый токен

catGeneratorCanceletionToken = new CancellationTokenSource();

var connection = GetConnection();

pbCatGenerator.Value = 0;

pbCatGenerator.Maximum = (int)udCount.Value;

var breeds = new Dictionary<string, TrackBar>

{

["Сибирская"] = tbSiberian,

["Бенгальская"] = tbBengalskaya,

["Сиамская"] = tbSiamskaya,

["Мейн-кун"] = tbMainKun,

["Египетская"] = tbEgipetskaya,

["Персидская"] = tbPersidkaya,

["Манчкин"] = tbManchkin,

["Шоколадный йорк"] = tbShocoladniyYork,

};

int totalBreed = breeds.Values.Sum(tb => tb.Value);

Dictionary<string, Range> breedsProbablity = new();

double previousProbabilty = 0;

foreach (var item in breeds)

{

double k = (double)breeds[item.Key].Value / totalBreed;

if (k > 0)

{

breedsProbablity[item.Key] = new Range

{

min = previousProbabilty,

max = previousProbabilty + k

};

previousProbabilty += k;

}

}

int count = (int)udCount.Value;

int maleFemale = (int)tbCatMaleFemale.Value;

var progress = new Progress<ProgressInfo>(progress =>

{

pbCatGenerator.Value = progress.value;

lblRealCatsCounts.Text = progress.info;

});

btnGenerateCats.Text = "Отмена";

await Task.Run(() =>

{

GenerateCats(

connection,

breedsProbablity,

count,

maleFemale,

progress,

catGeneratorCanceletionToken // передаем токен на отмену

);

});

//btnGenerateCats.Enabled = true;

btnGenerateCats.Text = "Запустить";

catGeneratorCanceletionToken.Dispose(); // по завершению генерации очищаем память под токен

catGeneratorCanceletionToken = null; // и сбрасываем его в null;

}

# 5 Код генерации локаций

## 5.1 Описание

Генератор локаций позволяет указать количество экземпляров локаций, которые будут добавлены в базу данных. Локация может иметь один из четырех уровней. Первый уровень соответствует зданию. Всего доступно пять типов зданий: Дом, Усадьба, Пансионат, Небоскреб, Коттедж. Генератор рандомно выбирает тип здания. Второй уровень соответствует этажу. Система позволяет указать количество этажей в здании. Третий уровень соответствует квартире. Система позволяет указать максимальное количество квартир на этаже. Четвертый уровень соответствует комнате в квартире. Система позволяет указать максимальное количество комнат в квартире. Также доступна возможность указать два особых типа комнат: ванная и балкон.

## 5.1 Код функции генерирующей здание

readonly string[] BUILDING\_TYPES = new[] { "Дом", "Усадьба", "Пансионат", "Небоскреб", "Коттедж" };

private void GenerateBuilding(

SqlConnection connection,

SqlTransaction transaction,

int floorsMin,

int floorMax,

int apMin,

int apMax,

int roomMin,

int roomMax

)

{

var maleAdjectives = this.adjectives.Where(x => x.gender == "муж").ToList();

var adjective = maleAdjectives.ElementAt(random.Next(0, maleAdjectives.Count()));

var buildingType = BUILDING\_TYPES.ElementAt(random.Next(0, BUILDING\_TYPES.Count()));

var buildingName = $"{adjective.word.Capitalize()} {buildingType}";

SqlCommand command = new($@"INSERT INTO Locations(title) OUTPUT inserted.id VALUES(@name)", connection);

command.Transaction = transaction;

command.Parameters.Add("@name", SqlDbType.Text);

command.Parameters["@name"].Value = buildingName;

int buildingId = (int)command.ExecuteScalar();

var floorsCount = random.Next(floorsMin, floorMax);

// у меня будет сквозная нумерация квартир, поэтому добавляю поле

var nextAppartamentNumber = 1;

for (var floor = 1; floor <= floorsCount; ++floor)

{

var floorName = $"{floor} этаж";

// добавляем OUTPUT inserted.id

command = new($@"INSERT INTO Locations(title, parent\_location\_id) OUTPUT inserted.id VALUES(@floorName, @buildingId)", connection);

command.Transaction = transaction;

command.Parameters.Add("@floorName", SqlDbType.Text);

command.Parameters["@floorName"].Value = floorName;

command.Parameters.Add("@buildingId", SqlDbType.Int);

command.Parameters["@buildingId"].Value = buildingId;

// заменяем command.ExecuteNonQuery();

int floorId = (int)command.ExecuteScalar();

// ну и тут все по аналогии

// сначала рандомно бросаем количество квартри

var appartamentsCount = random.Next(apMin, apMax);

for (var a = 1; a <= appartamentsCount; ++a)

{

// формируем имя квартиры

var appartamentName = $"{nextAppartamentNumber} квартира";

// увеличиваем номер квартиры в сквозной нумерации

nextAppartamentNumber++;

// ну и формируем запрос, передаваем

command = new($@"INSERT INTO Locations(title, parent\_location\_id) OUTPUT inserted.id VALUES(@appartamentName, @floorId)", connection);

command.Transaction = transaction;

command.Parameters.Add("@appartamentName", SqlDbType.Text);

command.Parameters["@appartamentName"].Value = appartamentName;

command.Parameters.Add("@floorId", SqlDbType.Int);

command.Parameters["@floorId"].Value = floorId;

// заменяем command.ExecuteNonQuery();

int appartamentId = (int)command.ExecuteScalar();

var roomsCount = random.Next(roomMin, roomMax);

for (var r = 1; r <= roomsCount; ++r)

{

var roomName = $"{r} комната";

command = new($@"INSERT INTO Locations(title, parent\_location\_id) OUTPUT inserted.id VALUES(@roomName, @appartamentId)", connection);

command.Transaction = transaction;

command.Parameters.Add("@roomName", SqlDbType.Text);

command.Parameters["@roomName"].Value = roomName;

command.Parameters.Add("@appartamentId", SqlDbType.Int);

command.Parameters["@appartamentId"].Value = appartamentId;

command.ExecuteNonQuery();

}

if (chkHasBaclocny.Checked)

{

var roomName = $"Балкон";

command = new($@"INSERT INTO Locations(title, parent\_location\_id) OUTPUT inserted.id VALUES(@roomName, @appartamentId)", connection);

command.Transaction = transaction;

command.Parameters.Add("@roomName", SqlDbType.Text);

command.Parameters["@roomName"].Value = roomName;

command.Parameters.Add("@appartamentId", SqlDbType.Int);

command.Parameters["@appartamentId"].Value = appartamentId;

command.ExecuteNonQuery();

}

if (chkHasBathroom.Checked)

{

var roomName = $"Ванная";

command = new($@"INSERT INTO Locations(title, parent\_location\_id) OUTPUT inserted.id VALUES(@roomName, @appartamentId)", connection);

command.Transaction = transaction;

command.Parameters.Add("@roomName", SqlDbType.Text);

command.Parameters["@roomName"].Value = roomName;

command.Parameters.Add("@appartamentId", SqlDbType.Int);

command.Parameters["@appartamentId"].Value = appartamentId;

command.ExecuteNonQuery();

}

Thread.Sleep(10);

}

}

}

## 5.3 Код функции генерирующей локации

CancellationTokenSource locationsGeneratorCanceletionToken = null;

private void GenerationLocations(

SqlConnection connection,

int count,

int floorsMin,

int floorMax,

int apMin,

int apMax,

int roomMin,

int roomMax,

bool useTransactions,

IProgress<ProgressInfo> progress,

CancellationTokenSource token

)

{

using (connection)

{

connection.Open();

var transaction = useTransactions ? connection.BeginTransaction() : null;

for (var i = 0; i < count; ++i)

{

GenerateBuilding(

connection,

transaction,

floorsMin, floorMax,

apMin, apMax,

roomMin, roomMax

);

if (token.IsCancellationRequested)

return;

var command = new SqlCommand($"SELECT count(\*) FROM Locations", connection);

command.Transaction = transaction;

var info = $"Количество локаций в транзакции {command.ExecuteScalar()}";

progress.Report(new ProgressInfo

{

value = i + 1,

info = info

});

}

}

}

## 5.4 Код реакции на нажатие кнопки сгенерировать локации

private async void btnGenerateLocations\_Click(object sender, EventArgs e)

{

if (locationsGeneratorCanceletionToken != null)

{

locationsGeneratorCanceletionToken.Cancel();

return;

}

locationsGeneratorCanceletionToken = new CancellationTokenSource();

btnGenerateLocations.Text = "Отмена";

var connection = GetConnection();

pbLocationsGenerator.Value = 0;

pbLocationsGenerator.Maximum = (int)udLocationsCount.Value;

int count = (int)udLocationsCount.Value;

int floorsMin = tbFloors.Minimum,

floorMax = tbFloors.Value + 1,

apMin = tbApartaments.Minimum, apMax = tbApartaments.Value + 1,

roomMin = tbRooms.Minimum, roomMax = tbRooms.Value + 1;

var progress = new Progress<ProgressInfo>(p =>

{

pbLocationsGenerator.Value = p.value;

lblLocationsRealCount.Text = p.info;

});

await Task.Run(() =>

{

GenerationLocations(

connection,

count,

floorsMin,

floorMax,

apMin,

apMax,

roomMin,

roomMax,

chkLocationsUseTransactions.Checked,

progress,

locationsGeneratorCanceletionToken

);

});

btnGenerateLocations.Text = "Запустить";

lblLocationsRealCount.Text = "";

locationsGeneratorCanceletionToken = null;

}

# 6 Статистика

## 6.1 Описание

Статистика автоматически запрашивается из базы данных по тику таймера. Статистика агрегирует данные базы и выдает информацию о:

* Общее количество кошек и котов
* Количество кошек и котов по породам
* Общее количество локаций
* Количество локаций по уровням

## 6.2 Код запроса статистики

private Statistics GetStatistics(SqlConnection connection)

{

var statistics = new Statistics();

using (connection)

{

connection.Open();

var command = new SqlCommand("SELECT count(\*) FROM cats", connection);

statistics.catsCount = (int)command.ExecuteScalar();

command = new SqlCommand("SELECT count(\*) FROM locations", connection);

statistics.locationsCount = (int)command.ExecuteScalar();

command = new SqlCommand("SELECT breed, count(\*) as count FROM cats GROUP BY breed", connection);

statistics.catsCountsByBreed = new Dictionary<string, int>();

using (var reader = command.ExecuteReader())

{

while (reader.Read())

{

if (!reader.IsDBNull("breed"))

{

statistics.catsCountsByBreed[reader.GetString("breed")] = reader.GetInt32("count");

}

}

}

command = new SqlCommand(@"

SELECT depth, count(\*) as count

FROM (

SELECT DISTINCT l1.id, l1.title,

CASE WHEN l2.id is not NULL THEN 0 ELSE 1 END

+ CASE WHEN l3.id is not NULL THEN 0 ELSE 1 END

+ CASE WHEN l4.id is not NULL THEN 0 ELSE 1 END + 1 as depth

FROM Locations l1

LEFT JOIN Locations l2 ON l2.parent\_location\_id = l1.Id

LEFT JOIN Locations l3 ON l3.parent\_location\_id = l2.Id

LEFT JOIN Locations l4 ON l4.parent\_location\_id = l3.Id

) t

GROUP BY depth", connection);

statistics.locationCountsByDepth = new ();

using (var reader = command.ExecuteReader())

{

while (reader.Read())

{

statistics.locationCountsByDepth[reader.GetInt32("depth")] = reader.GetInt32("count");

}

}

}

return statistics;

}

## 6.3 Код тика таймера

private async void timer1\_Tick(object sender, EventArgs e)

{

var connection = GetConnection();

var statistics = await Task.Run(() =>

{

return GetStatistics(connection);

});

var breedsInfo = String.Join(

"\n",

statistics.catsCountsByBreed

.Select(x => x.Key)

.OrderBy(x => x)

.Select(x => $" - {x}: {statistics.catsCountsByBreed[x]}")

);

var locationsInfo = String.Join(

"\n",

statistics.locationCountsByDepth

.Select(x => x.Key)

.OrderBy(x => x)

.Select(x => {

var key = x switch {

1 => "домов",

2 => "этажей",

3 => "квартир",

4 => "комнат",

\_ => ""

};

return $" - {key}: {statistics.locationCountsByDepth[x]}";

})

);

lblStatistics.Text = $@"

Кошек и котов: {statistics.catsCount},

{breedsInfo}

-=-=-=-=-=-=-=-=-=-=-=-

Локаций: {statistics.locationsCount}

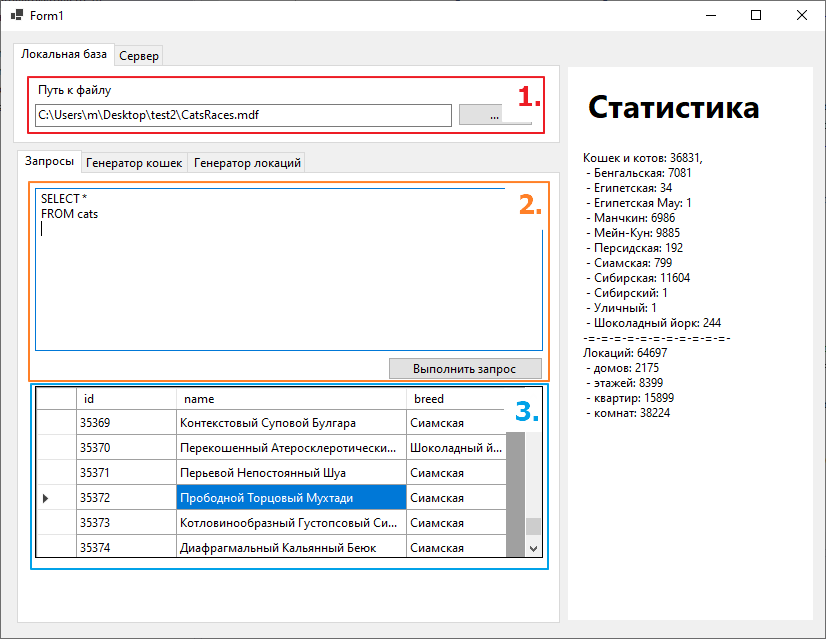
{locationsInfo}

".Trim();

}

# 7 Описание работы интерфейса

После запуска приложения есть возможность указать путь к файлу с базой (см. рисунок 7.1(1)), запустить произвольный запрос (см. рисунок 7.1(2)) и получить данные в табличном представлении (см. рисунок 7.1(3)).

  
Рисунок 7.1

На вкладке «Генератор кошек» можно выбрать соотношение котов/кошек (см. рисунок 7.2(1)), отдельно выделена группа трекбаров позволяющая указать долю пород при генерации (см. рисунок 7.2(2)). Есть отдельный контролер для указания общего количества кошек, которые будут сгенерированы (см. рисунок 7.2(3)). Есть чекбокс (см. рисунок 7.2(4)), который позволяет указать следует ли выполнять генерацию в транзакции или нет. Также в правом нижнем углу есть кнопка для запуска генератора (см. рисунок 7.2(5)).

Генератор работает асинхроно и не блокирует интерфейс. На время генерации кнопка меняет свой текст с «Запустить» на «Отмена» и изменяет свое поведение. Клик на кнопку позволяет приостановить генерацию (см. рисунок 7.3).

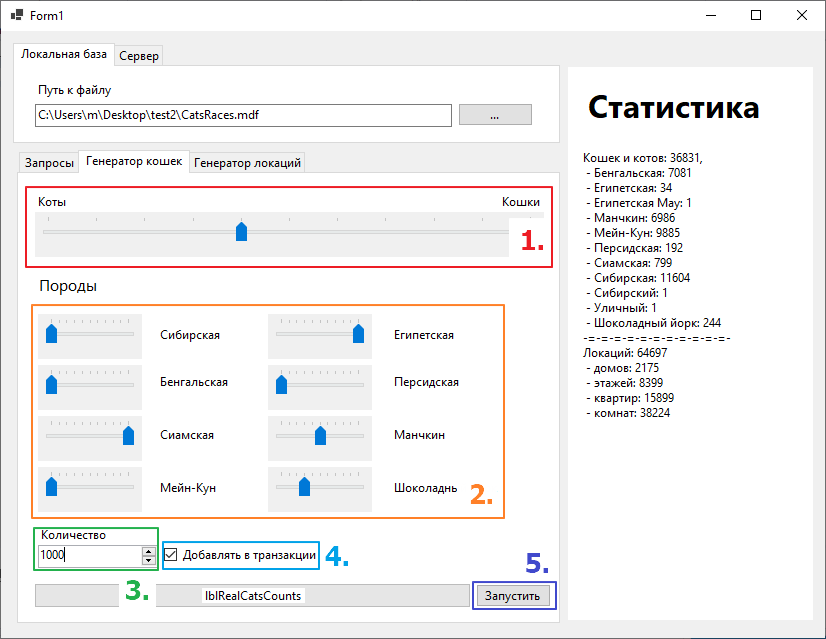


Рисунок 7.2



Рисунок 7.3

На вкладке «Генератор локаций» можно сконфигурировать генератор локаций. Указать максимальную этажность зданий (см. рисунок 7.4(1)), настроить количество квартир на этаже (см. рисунок 7.4(2)), указать количество комнат в квартире (см. рисунок 7.4(3)). Выделены два отдельных чекбокса повзоляющие указать наличие ванной и балкона в квартире (см. рисунок 7.4(4)). Есть отдельный контролер для указания общего количества кошек, которые будут сгенерированы (см. рисунок 7.4(5)). Имеется чекбокс (см. рисунок 7.4(6)), который позволяет указать следует ли выполнять генерацию в транзакции или нет. Также в правом нижнем углу есть кнопка для запуска генератора (см. рисунок 7.4(7)).

Генератор работает асинхроно и не блокирует интерфейс. На время генерации кнопка меняет свой текст и изменяет свое поведение. Клик на кнопку позволяет приостановить генерацию (см. рисунок 7.5).

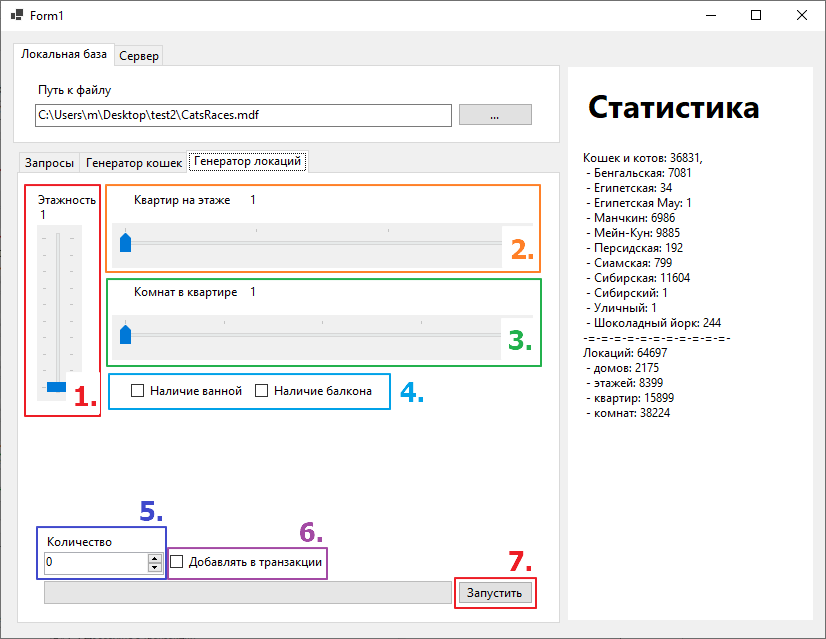


Рисунок 7.4

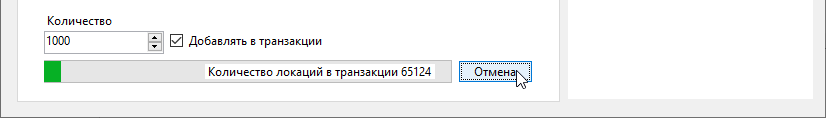
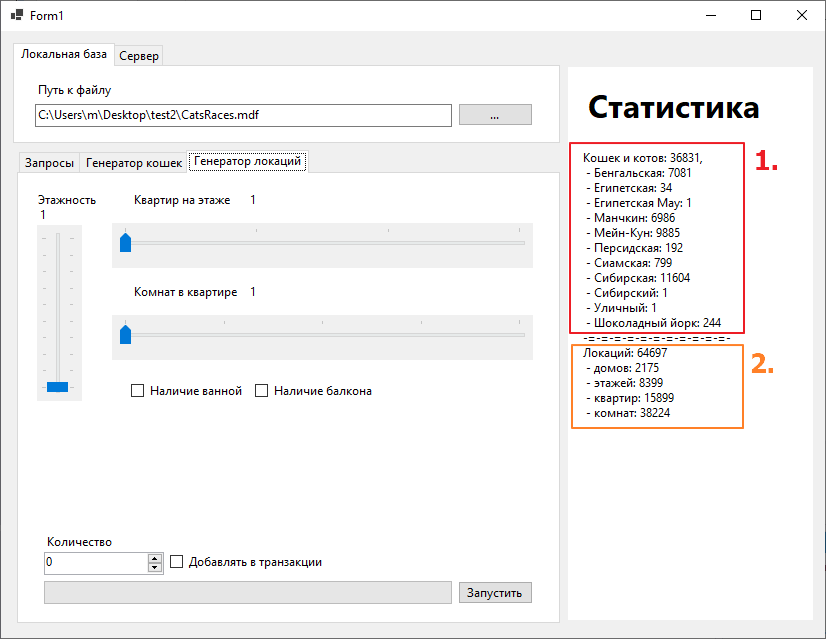


Рисунок 7.5

Справа на форме выводится статистика. В статистике отображается информация об общем количестве кошек и котов, а также информация о количестве особей по породам (см. рисунок 7.6(1)). Также в статистике присутсвет информация об общем количестве локаций, а также дополнительная информация о количестве локаций в зависимости от уровня (см. рисунок 7.6(2)).

Обновление статистики происходит асинхроно по таймеру раз в 1 секунду.

 Рисунок 7.6

# Заключение

В рамках курсовой работы было разработано приложение на языке C# взаимодействующее с СУБД Microsoft SQL Server, позволяющее эмитировать процесс управления данными в условиях высоконагруженных систем. Разработанная система позволяет:

* Генерировать данные для таблиц в БД
* Конфигурировать генератор данных
* Выводить оперативную статистику по таблицам в БД
* Выполнять несколько параллельных запросов
* Оборачивать выполнение запросов к базе данных в транзакции

Приложение было всесторонне протестировано и отлажено. Все поставленные в рамках курсовой работы задачи были выполнены.

В рамках выполнения данной работы был получен обширный спектр сведений об управлении большими объемами данных и обработки конкурентных запросов. А также освоен механизм работы с транзакциями и асинхронными функциями в высокоуровневом языке программирования C#.

# Список использованных источников

1. Грубер М. Понимание SQL / Мартин Грабер. – Москва, 1993. – 291 c.
2. Петкович Д. Microsoft® SQL Server™ 2012. Руководство для начинающих: Пер с англ. / Душан Петкович. – СПБ.: БХВ-Петербург, 2013. – 816 с.
3. Управление Данными [Электронный ресурс] // kaboom.tealeaf.su : чаинка, 2021. URL: http://kaboom.tealeaf.su (дата обращения: 01.12.2021).
4. Молинаро Э. SQL Cookbook / Энтони Молинаро. – O'Reilly Media, 2005 – 596 с.