министерство НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
Российской федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Тюменский государственный университет»

институт математики и компьютерных наук

Кафедра программного обеспечения

Курсовая работа по направлению

«Математическое обеспечение и администрирование

информационных систем»

на тему «Разработка ПО для резервного копирования и восстановления баз данных MS SQL Server»

Выполнил: студент 3 курса

22 МОиАИС 184-2 группы

Поляков И.А.

Научный руководитель:

доцент кафедры ПО

Ялдыгин В.Б.

Тюмень 2021

Оглавление

[ВВЕДЕНИЕ 3](#_Toc88760151)

[ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 5](#_Toc88760152)

[1.1. Основные сведения о резервном копировании и восстановлении баз данных MS SQL Server 5](#_Toc88760153)

[1.2. Типы резервного копирования SQL Server 6](#_Toc88760154)

[1.2.1. Полное копирование базы данных 6](#_Toc88760155)

[1.2.2. Дифференциальное резервное копирование 9](#_Toc88760156)

[1.2.3. Резервное копирование протокола транзакции 10](#_Toc88760157)

[1.2.4. Резервное копирование файла или файловой группы. Частичная резервная копия 10](#_Toc88760158)

[ГЛАВА 2. СОЗДАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ РЕЗЕРВНОГО КОПИРОВАНИЕ И ВОССТАНОВЛЕНИЯ БАЗЫ ДАННЫХ 11](#_Toc88760159)

[2.1 Используемые технологии 11](#_Toc88760160)

[2.2 Общие сведения об SMO 11](#_Toc88760161)

[2.3 Классы модели объектов SMO 12](#_Toc88760162)

[2.4 Функции SMO 13](#_Toc88760163)

[2.5 Реализованные классы и методы 15](#_Toc88760164)

[ГЛАВА 3. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ И ОПИСАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 18](#_Toc88760165)

[ЗАКЛЮЧЕНИЕ 21](#_Toc88760166)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 22](#_Toc88760167)

[Приложение 1 23](#_Toc88760168)

[Приложение 2 27](#_Toc88760169)

[Приложение 3 29](#_Toc88760170)

# ВВЕДЕНИЕ

Рост ценности данных для работы компаний и сопутствующий ему рост объемов этих данных увеличивает важность систем резервного копирования и восстановления для поддержания непрерывности бизнеса. В условиях пандемии и перехода большого количества сотрудников на удаленную работу сохранность корпоративных данных и возможность их оперативного восстановления стали еще более актуальными. Соответственно, сегмент средств резервного копирования растет даже в условиях падения ИТ-рынка в целом. А появление новых задач приводит к появлению новых лидеров.

У вас может быть отличный компьютер или внешний жесткий диск, но однажды они износятся, и вы можете потерять свои данные. Такова природа любого аппаратного обеспечения. Люди, занимающиеся ремонтом компьютеров, может быть, смогут спасти ваши данные, а может быть и нет. Это всегда рискованно, если вы не сделаете резервную копию своих данных.

Хуже того, интернет несет в себе множество потенциальных угроз для ваших данных. Такие агенты, как вирусы и трояны, не просто крадут ваши данные, во многих случаях они просто стирают их. Также существует угроза вымогателей. Это когда хакер помещает вирус на ваш компьютер, который шифрует ваши данные, что делает его бесполезным для вас устройством. Возможно, вам придется заплатить выкуп, чтобы хакер смог расшифровать ваши данные, конечно, без гарантии того, что он это сделает. Но если у вас есть текущее резервное копирование данных, то это не вызовет беспокойства. Вы сможете просто стереть свой жесткий диск и восстановить его последнюю резервную копию.

Из этого можно сделать вывод, что хранение данных, должно обеспечивать их целостность. Большинство данных хранятся в базах, так как это является удобным способом взаимодействия с ними. Одной из основных технологий баз данных является SQL Server, который является реляционной системой управления базами данных (РСУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов — Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия, SQL Server конкурирует с другими СУБД в этом сегменте рынка. Для сохранения целостности баз данных используют такие технологии как их резервное копирование и восстановление.

Цель работы: создать систему резервного копирования, позволяющую восстанавливать и резервировать данные, а также автоматизировать этот процесс.

Задачи:

1. Изучить основные задачи и методы резервного копирования и восстановления баз данных.
2. Разработать ПО для выполнения резервного копирования и восстановления.
3. Составить руководство пользователя.

# ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

## 1.1. Основные сведения о резервном копировании и восстановлении баз данных MS SQL Server

Резервное копирование базы данных — это процесс загрузки данных (из базы данных, журнала транзакций или файла) на устройства резервного копирования, которые создаются и обслуживаются системой. Устройство резервного копирования может быть дисковым файлом или магнитной лентой. Компонент Database Engine обеспечивает статическое и динамическое резервное копирование.

Статическое резервное копирование означает, что во время процесса резервного копирования только один активный сеанс, поддерживаемый системой, является сеансом, создающим резервную копию. Другими словами, во время копирования никакие пользовательские процессы не допускаются.

Динамическое резервное копирование означает, что копирование базы данных может выполняться без остановки сервера базы данных, удаления пользователей и даже без закрытия файлов (пользователи даже не узнают, что идет процесс резервного копирования).

К одной из важнейших задач администратора базы данных можно отнести постоянную защиту целостности баз данных, а также поддержку возможности быстрого восстановления в случае сбоя. Именно из-за этого тактика резервного копирования и восстановления важны. Это поможет избежать чрезвычайных ситуаций.

Еще одна важная обязанность администратора – это доступность базы данных в тот момент, когда это необходимо. Ответственность за восстановление базы данных в случае, если она была удалена, повреждена или сброшена, лежит на администраторе базы данных.

Человек, отвечающий за базу данных, должен быть готов к её аварийному восстановлению. Это достигается путем постоянного тестирования резервного копирования и восстановления SQL Server. В таком случае можно избежать потери данных. Также к обязанностям администратора базы данных входит и её защита от различных сбоев.

## 1.2. Типы резервного копирования SQL Server

Компонент Database Engine предоставляет четыре различных метода резервного копирования:

1. Полное копирование базы данных;
2. Дифференцированное резервное копирование;
3. Резервное копирование протокола транзакции;
4. Резервное копирование файла или файловой группы. Частичная резервная копия.

## 1.2.1. Полное копирование базы данных

Полное резервное копирование делает копию всей базы данных, включая все объекты и данные системных таблиц. Полная резервная копия не будет усекать (truncate) журнал транзакций. Это основной тип резервных копий, который требуется выполнять перед другими типами резервных копий.

Полную резервную копию вы можете восстановить за 1 шаг, так как она не требует других дифференциальных/инкрементальных копий.

Если модель восстановления базы SQL данных установлена как “Полная”, то при восстановлении бекапа вы можете указать параметр “**STOPAT**”, где указывается время (до секунды) на котором нужно остановить восстановление данных. Например, сотрудник внёс некорректные данные в 14:46:07, с помощью параметра STOPAT вы можете восстановить данные на момент 14:46:06

Все типы резервных копий можно выполнять с помощью двух операторов Transact-SQL:

1. backup database;
2. backup log.

Компонент Database Engine позволяет копировать базы данных, журналы транзакций и файлы на устройства резервного копирования, такие как:

1. Диск;

2. Магнитная лента.

Дисковые файлы чаще всего используются для хранения резервных копий. Диск с копией может быть расположен на локальном жестком диске на сервере или на удаленном диске в общей сетевой папке. Компонент Database Engine позволяет добавлять новые копии в файл, который уже содержит копии той же или другой базы данных. Когда вы добавляете новый набор копий на существующий носитель, предыдущее содержимое носителя остается нетронутым, и новая копия записывается после последней копии на этом носителе. (Набор копий включает все сохраненные данные объекта, которые вы выбираете для копирования.) По умолчанию компонент Database Engine всегда добавляет новые копии в файлы на диске.

Магнитная лента обычно используется в качестве устройства хранения резервных копий так же, как и дисковый накопитель. Однако при резервном копировании на ленту ленточное устройство должно быть локально подключено к системе. Преимущество ленточных устройств перед дисковыми - простота администрирования и эксплуатации.

Сценарий восстановления — SQL Server — это процесс восстановления данных из одной или нескольких резервных копий и возврата базы данных в исходное состояние. Поддерживаемые сценарии восстановления зависят от модели восстановления базы данных и выпуска SQL Server.

Для большинства сценариев восстановления необходимо применить резервную копию журнала транзакций и позволить ядру СУБД SQL Server запустить процесс восстановления, чтобы перевести базу данных в оперативный режим. Восстановление — это процесс, используемый SQL Server для запуска каждой базы данных в транзакционно согласованном (чистом) состоянии.

В случае аварийного переключения или другого неясного завершения работы базы данных могут оставаться в состоянии, в котором некоторые изменения не были записаны из буферного кеша в файлы данных, и в файлах данных могут быть некоторые изменения, вызванные незавершенными транзакциями. Когда вы запускаете экземпляр SQL Server, выполняется трехэтапное восстановление каждой базы данных на основе самой последней контрольной точки в базе данных:

1. На этапе анализа анализируется журнал транзакций для определения последней контрольной точки и создается таблица грязных страниц (DTP) и таблица активных транзакций (TAT). TGS содержит записи о страницах, которые были грязными при выключении базы данных. TAT содержит записи транзакций, которые все еще были активны, когда база данных была аварийно завершена.
2. На этапе повтора повторяются все зарегистрированные изменения, которые, возможно, не были записаны в файлы данных, когда база данных была закрыта. Минимальный порядковый номер журнала (minLSN), необходимый для успешного восстановления всей базы данных, находится в TGS и отмечает начало операций восстановления, необходимых для всех грязных страниц. На этом этапе компонент SQL Server Database Engine записывает на диск все грязные страницы, принадлежащие зафиксированным транзакциям.
3. На этапе отката выполняется откат ожидающих транзакций, обнаруженных в TAT, чтобы гарантировать сохранение целостности базы данных. После отката база данных переводится в оперативный режим, и больше нельзя применять резервные копии журнала транзакций.

Цель полного восстановления – восстановить всю базу данных. База данных отключена во время периода восстановления. Перед тем, как какая-либо часть базы данных перейдет в оперативный режим, все данные восстанавливаются до точки согласованности, когда все части базы данных находятся в один и тот же момент времени и отсутствуют незавершенные транзакции.

В режиме полного восстановления после восстановления резервных копий данных необходимо восстановить все последующие резервные копии журналов транзакций, а затем и саму базу данных. База данных может быть восстановлена до определенной точки восстановления в одной из этих резервных копий журнала. Этой точкой восстановления может быть указанная дата и время, помеченная транзакция или порядковый номер журнала (LSN).

При восстановлении базы данных, особенно при использовании модели восстановления с полным или неполным протоколированием, используйте одну последовательность восстановления. Последовательность восстановления состоит из одной или нескольких операций восстановления, которые перемещают данные в одну или несколько фаз восстановления.

## 1.2.2. Дифференциальное резервное копирование

Дифференциальное или разностное резервное копирование — это копирование только тех данных, которые появились с момента последней полной резервной копии.

Данный тип резервного копирования используют совместно с полной резервной копией, так как для восстановления дифференциальной копии необходима полная резервная копия.

Обычно при использовании разностного резервного копирования используют план по типу “полное раз в N дней, дифференциальное каждые N часов”. Если ежедневный оборот данных достаточно высокий, то данный тип резервных копий может быть неудобен в применении, так как копии будут весить довольно много.

Например, если полная резервная копия весит 300 GB, а дифференциальная спустя час работы 5 GB, то спустя сутки это будет 120 GB, что делает использование данного типа копий нерациональным.

## 1.2.3. Резервное копирование протокола транзакции

Резервное копирования журнала транзакций копирует все транзакции, которые произошли с момента последнего резервного копирования, а затем урезает журнал транзакций для освобождения дискового пространства.

Восстанавливая журнал транзакций, вы также можете указать параметр STOPAT, как и в восстановлении полной резервной копии.

Этот тип бекапа является инкрементальным, поэтому для восстановления базы данных вам потребуется вся цепочка резервных копий: Полная и все последующие инкрементальные журнала транзакций.

## 1.2.4. Резервное копирование файла или файловой группы. Частичная резервная копия

Partial backup (частичное резервное копирование) – это тип резервной копии, который используется для снятия копии с read-only файловых групп. Используется редко.

Резервное копирование файлов или файловых групп используется для снятия резервных копий определенных файлов или файловых групп.

# ГЛАВА 2. СОЗДАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ РЕЗЕРВНОГО КОПИРОВАНИЕ И ВОССТАНОВЛЕНИЯ БАЗЫ ДАННЫХ

## 2.1 Используемые технологии

При создании программного обеспечения для резервного копирования и восстановления базы данных SQL Server, использовалась среда разработки Microsoft Visual Studio 2022, язык программирования C# на Windows Forms.

## 2.2 Общие сведения об SMO

SQL Server Управляющие объекты (SMO) — это объекты, предназначенные для программного управления Microsoft SQL Server. Объекты SMO можно использовать для создания специализированных приложений управления SQL Server. Хотя среда SQL Server Management Studio является мощным и универсальным приложением для управления SQL Server, иногда удобнее работать с приложением SMO.

К примеру, для удовлетворения потребностей новых пользователей, а также с целью сокращения издержек на подготовку персонала может возникнуть необходимость упрощения пользовательских приложений, выполняющих задачи управления SQL Server. Может возникнуть необходимость в создании специализированных баз данных SQL Server или в создании приложения для формирования индексов и наблюдения за их эффективностью. Кроме того, приложение SMO можно использовать для бесшовной интеграции аппаратных или программных компонентов от независимых поставщиков в приложение управления базами данных.

Поскольку модель объектов SMO совместима с SQL Server 2005 (9.x) и более поздними версиями, облегчается работа с продуктами различных версий.

## 2.3 Классы модели объектов SMO

Классы модели объектов SMO подразделяются на две категории: классы экземпляров и служебные классы.

**Классы экземпляров**

Классы экземпляров представляют объекты SQL Server, такие как серверы, базы данных, таблицы, триггеры и хранимые процедуры. Класс [ServerConnection](https://docs.microsoft.com/ru-ru/dotnet/api/microsoft.sqlserver.management.common.serverconnection) используется для установления соединения с экземпляром SQL Server и для управления режимом сбора направляемых ему команд.

Объекты экземпляра модели SMO образуют иерархию, которая представляет иерархию сервера баз данных. Вверху размещаются экземпляры SQL Server, под ними располагаются базы данных, а далее — таблицы, столбцы, триггеры и т. д. Если логика допускает возможность существования связи «один родитель ко многим потомкам», например в таблице с одним или несколькими столбцами, тогда потомок представляется коллекцией объектов. В других случаях потомок представляется одним объектом.

**Служебные классы**

Служебные классы — это группа объектов, созданных явным образом для выполнения конкретных задач. В соответствии со своими функциями они разделяются на различные иерархии объектов.

Класс Transfer. Используется для передачи другой базе данных схемы и данных.

Классы Backup и Restore. Используются для создания резервной копии и восстановления баз данных.

Класс Scripter. Используется для формирования файлов скриптов, предназначенных для повторного создания объектов и их зависимостей.

## 2.4 Функции SMO

**Оптимизированная производительность**

Архитектура объектов SMO эффективна с точки зрения памяти, поскольку объекты создаются только частично, а на сервере запрашиваются минимальные сведения о свойствах. Полное создание объектов откладывается до того времени, когда выполняется явная ссылка на объект. Объект полностью создается тогда, когда требуется свойство, не входящее в набор изначально полученных свойств, или когда вызывается метод, требующий такое свойство. Переход от частично полностью созданных объектов к полностью созданным незаметен для пользователя. Кроме того, некоторые свойства, использующие большие объемы памяти, так и не считываются — если только не выполняется явное обращение к подобному свойству. Примером сказанному может служить свойство [Size](https://docs.microsoft.com/ru-ru/dotnet/api/microsoft.sqlserver.management.smo.database.size) свойства объекта [Database](https://docs.microsoft.com/ru-ru/dotnet/api/microsoft.sqlserver.management.smo.database). Однако при частичном создании объектов возникает большее количество обменов данными через сеть, и потому с точки зрения производительности этот метод не обязательно будет наиболее эффективным для приложения.

Созданием экземпляров объектов можно управлять в соответствии с особенностями системной среды. При использовании метода отложенного создания экземпляров объем памяти, необходимый для работы приложения, сводится к минимуму, но на этапе обращения к свойствам может возникнуть большое число запросов к серверу.

Классы экземпляров (объекты, представляющие реальные объекты баз данных) могут существовать на трех уровнях создания. Эти уровни: минимальный экземпляр (лишь минимально необходимые свойства считываются в одном блоке), частично созданный экземпляр (все свойства, использующие относительно большие объемы памяти, считываются в одном блоке) и полностью созданный экземпляр. Традиционные состояния создания экземпляров — несозданный и полностью созданный. Состояние «частично созданный» повышает эффективность, так как частично созданный объект не содержит значений для полного набора свойств объекта. Состояние «частично созданный» применяется по умолчанию для объектов, на которые нет непосредственных ссылок. При обращении к одному из этих свойств возникает ошибка, которая инициирует создание полного экземпляра объекта.

**Выполнение после сбора**

Обычным методом выполнения является непосредственное выполнение. Инструкции передаются экземпляру SQL Server непосредственно по получении. Выполнение после сбора является альтернативным методом.

Выполнение после сбора дает возможность собирать пакеты Transact-SQL, которые обычно выполняются. В результате программист модели объектов SMO может отложить скрипт, сохранить его для выполнения в более поздний период или выполнить предварительный просмотр скрипта для конечного пользователя. Так, инструкции **create database**, **create table** и **create index** могут быть переданы в одном пакете и затем выполнены как три последовательных шага. Этой функцией управляет пользователь с помощью объекта [Server](https://docs.microsoft.com/ru-ru/dotnet/api/microsoft.sqlserver.management.smo.server.-ctor).

**Поставщик WMI**

Объекты поставщика WMI помещаются в объекты SMO. В результате программист модели SMO получает простую модель объектов, весьма напоминающую классы SMO. Однако в этом случае программист не должен понимать модель программирования, представленную пространством имен, и особенности организации поставщика WMI SQL Server. Поставщик WMI позволяет конфигурировать службы SQL Server, псевдонимы, а также сетевые библиотеки клиентов и серверов.

**Создание скриптов**

В модели объектов SMO средства для работы со скриптами были улучшены и переданы в класс **Scripter**. С помощью класса **Scripter** разработчик может находить зависимости, выявлять связи между объектами и осуществлять манипуляции с иерархией зависимостей. Главным объектом, обеспечивающим работу со скриптами, является объект **Scripter**. Существует несколько поддерживающих объектов, которые осуществляют обработку зависимостей, а также реагируют на события состояния и на события ошибок.

Объект **Scripter** поддерживает следующие дополнительные параметры создания скриптов:

простой однофазный метод (скрипт создается за один этап);

Продвинутые 3-фазные сценарии (создание скрипта выполняется в три этапа; обнаружение зависимостей, создание списка, создание скрипта)

двусторонний поиск зависимостей (позволяет выявлять зависимости, или зависимые элементы);

реагирование на события состояния;

реагирование на события ошибок.

**Уникальные имена ресурсов**

Ключевая концепция при использовании библиотеки объектов SMO — уникальные имена ресурсов (URN). Синтаксис URN напоминает синтаксис XPath. XPath — это путь иерархии, используемый для указания объекта, в котором каждый уровень имеет квалификаторы и функции. В модели объектов SMO URN имеет два элемента — путь, а также именование атрибутов, обладающее ограниченными функциональными возможностями. Путь используется для указания местоположения объекта, тогда как именование атрибутов дает возможность осуществлять частичную фильтрацию.

## 2.5 Реализованные классы и методы

При разработке приложения были реализованы следующие классы:  
Form1, Form2, dataBase.

Класс Form1 (Приложение 1):

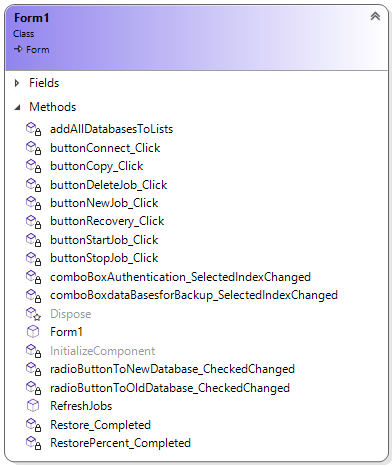


Диаграмма 1. Класс Form1

Методы класса Form1:

* public Form1() – Конструктор для инициализации form2 класса Form2 и заполнения списка серверов и методом аутентификации.
* void addAllDatabasesToLists() – Метод для заполнения списка баз данных сервера.
* private void buttonConnect\_Click(object sender, EventArgs e)- метод подключения к серверу.
* private void buttonCopy\_Click(object sender, EventArgs e)- метод создания бекапа базы данных.
* private void RestorePercent\_Completed(object sender, PercentCompleteEventArgs e) – метод для отображения прогресса восстановления.
* void Restore\_Completed(object sender, ServerMessageEventArgs e)- метод для отлова ошибки в случае некорректного восстановления.
* private void buttonRecovery\_Click(object sender, EventArgs e)- метод восстановления базы данных.
* private void comboBoxdataBasesforBackup\_SelectedIndexChanged(object sender, EventArgs e)- метод для вставки названия выбранной базы данных.
* private void radioButtonToOldDatabase\_CheckedChanged(object sender, EventArgs e)- метод контроля восстановления в новую или существующую базу данных.
* private void radioButtonToNewDatabase\_CheckedChanged(object sender, EventArgs e)- метод контроля восстановления в новую или существующую базу данных.
* private void comboBoxAuthentication\_SelectedIndexChanged(object sender, EventArgs e)-метод для контроля параметров аутентификации.
* public void RefreshJobs()- метод обновления существующих задач.
* private void buttonNewJob\_Click(object sender, EventArgs e)- метод вызова модального окна для создания новой задачи.
* private void buttonDeleteJob\_Click(object sender, EventArgs e)- метод удаления задачи
* private void buttonStartJob\_Click(object sender, EventArgs e)- метод старта задачи.
* private void buttonStopJob\_Click(object sender, EventArgs e)- метод остановки задачи.

Класс Form2 (Приложение 2):

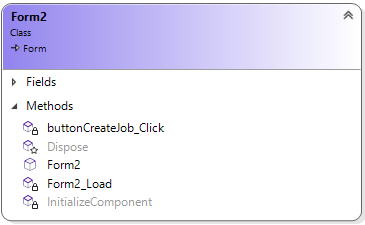


Диаграмма 2. Класс Form2

Методы класса Form1:

* public Form2()- конструктор для инициализации компонентов form2.
* private void Form2\_Load(object sender, EventArgs e)- метод загрузки form2, используется для заполнения списка базы данных при новом открытии.
* private void buttonCreateJob\_Click(object sender, EventArgs e)- метод создания задачи.

Класс DataBase (Приложение 3):

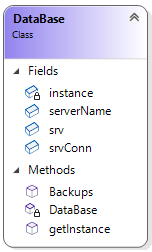


Диаграмма 3. Класс DataBase

Методы класса DataBase:

* private DataBase() { }- приватный конструктор.
* public static DataBase getInstance()- создание или возврат существующего объекта (шаблон одиночка).
* public string Backups(string NameBase, string Directory)- метод создания выражения команды бекапа.

# ГЛАВА 3. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ И ОПИСАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Для графической составляющей программного обеспечения использовался механизм Windows Forms. На Рисунке 1 представлено основное окно программного обеспечения.

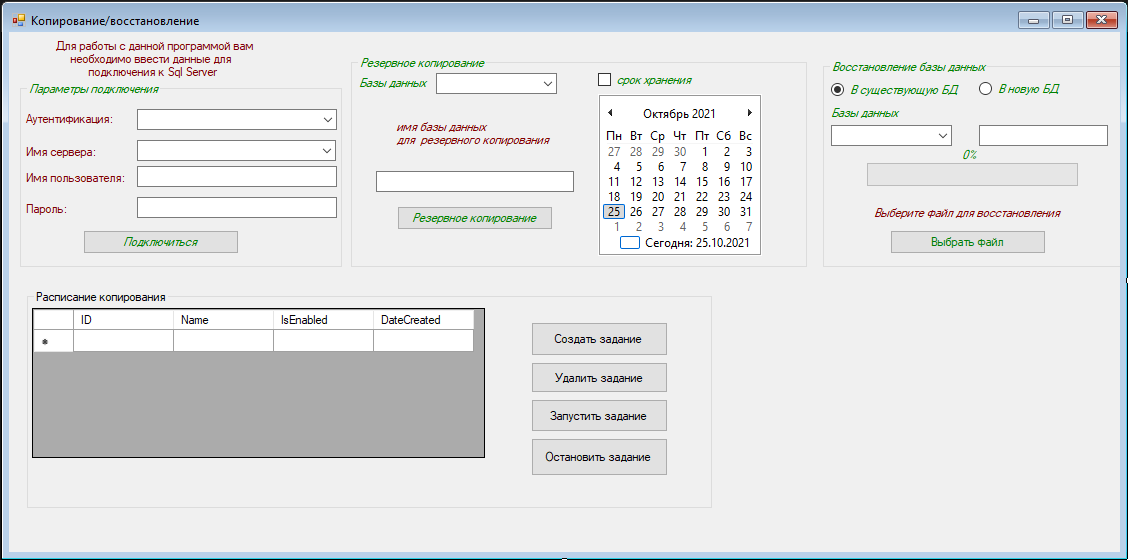


Рисунок 1. Основная форма программы

Данная форма делится на четыре блока:

1. Блок подключения к серверу.

Нужно выбрать сервер и вариант подключения профиля для управления базами данных

1. Блок резервного копирования.

База данных для копирования выбирается из списка баз данных или вводится вручную, возможно указать срок хранения бекапа.

1. Блок восстановления базы данных.

Два варианта восстановления, в новую или существую базу данных,

во время выполнения отображается прогресс.

1. Блок установки задач для бекапов баз данных.

Для управления существующими задачами выделяется строка задачи и выбирается вариант работы с ней, при выборе создания задачи открывается новое окно с настройками (Рисунок 2)

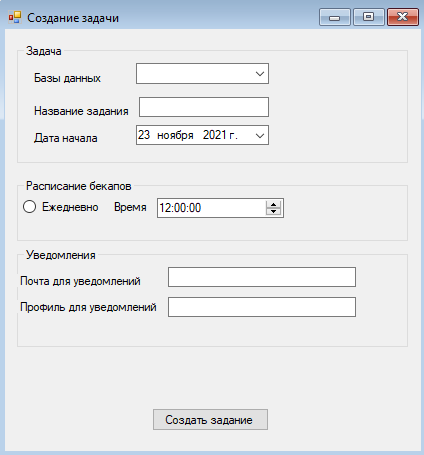


Рисунок 2. Форма установки задачи

Выбор базы данных и имени задачи, в расписании доступен выбор ежедневных бекапов или однократных, при каждом выполнении задачи на указанную почту высылается отчет оставшимся местом на диске, если почта не указана, то используется почта по умолчанию, как и профиль SSMS.

# ЗАКЛЮЧЕНИЕ

В ходе данной работы были рассмотрены основные факторы и алгоритмы, позволяющие провести резервное копирования и восстановление базы данных SQL Server. Построена структурная схема программы, по которой было реализовано программное обеспечение, позволяющее совершать резервное копирование и восстановление базы данных, не использую среду Microsoft SQL Server Management Studio.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. рынок решений резервного копирования 2020

https://www.cnews.ru/reviews/rynok\_reshenij\_rezervnogo\_kopirovaniya

1. Резервное копирование и восстановление баз данных SQL Server. <https://docs.microsoft.com/ru-ru/sql/relational-databases/backup-restore/back-up-and-restore-of-sql-server-databases?view=sql-server-ver15>
2. Резервное копирование и восстановление баз данных MSSQL.

https://winitpro.ru/index.php/2020/02/26/backup-i-vosstanovlenie-baz-dannyx-mssql/

1. Общие сведения об SMO.

https://docs.microsoft.com/ru-ru/sql/relational-databases/server-management-objects-smo/overview-smo?view=sql-server-ver15

1. Backup and Restore SQL Server databases programmatically with SMO. <https://www.mssqltips.com/sqlservertip/1849/backup-and-restore-sql-server-databases-programmatically-with-smo/>
2. Microsoft SQL Server. https://searchdatamanagement.techtarget.com/definition/SQL-Server
3. Агент SQL Server.

h<ttps://docs.microsoft.com/ru-ru/sql/ssms/agent/sql-server-agent?view=sql-s>erver-ver15

1. Создание задания.

<https://docs.microsoft.com/ru-ru/sql/ssms/agent/create-a-job?view=sql-server-ver15>

1. Запрос для отправки на почту.

<https://www.sqlshack.com/email-sql-query-results-smartphone-using-sp_send_dbmail-stored-procedure/>

# Приложение 1

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

using Microsoft.SqlServer.Management.Smo;

using Microsoft.SqlServer.Management.Smo.Agent;

using Microsoft.SqlServer.Management.Smo.Mail;

using Microsoft.SqlServer.Management.Common;

using MimeKit;

using MailKit.Net.Smtp;

namespace databaseBackup

{

public partial class Form1 : Form

{

Form2 form2;

DataBase dtb = DataBase.getInstance();

public Form1()

{

InitializeComponent();

comboBoxNameServer.SelectedItem = comboBoxNameServer.Items[0];

comboBoxAuthentication.SelectedItem = comboBoxAuthentication.Items[0];

form2 = new Form2();

}

void addAllDatabasesToLists()

{

comboBoxdataBasesforBackup.Items.Clear();

comboBoxdataBasesforRestore.Items.Clear();

foreach (Database item in dtb.srv.Databases)

{

comboBoxdataBasesforBackup.Items.Add(item.Name);

comboBoxdataBasesforRestore.Items.Add(item.Name);

}

}

private void buttonConnect\_Click(object sender, EventArgs e)

{

try

{

string Server = comboBoxNameServer.Text;

dtb.serverName = Server;

if (comboBoxAuthentication.SelectedIndex == 0)

{

dtb.srv = new Server(Server);

}

else

{

string NameUser = textBoxNameUser.Text;

string Pass = textBoxPass.Text;

dtb.srvConn = new ServerConnection(Server, NameUser, Pass);

dtb.srv = new Server(dtb.srvConn);

}

}

catch

{

MessageBox.Show("Ошибка соединения");

}

MessageBox.Show("Соединение установлено");

addAllDatabasesToLists();

buttonNewTask.Enabled = true;

RefreshJobs();

}

private void buttonCopy\_Click(object sender, EventArgs e)

{

string databaseName = textBoxNameBase.Text;

var bk = new Backup();

bk.Action = BackupActionType.Database;

bk.BackupSetDescription = "Full backup testdb";

bk.BackupSetName = "full backup";

if (comboBoxdataBasesforBackup.SelectedItem != null)

bk.Database = comboBoxdataBasesforBackup.SelectedItem.ToString();

bk.Incremental = false;

if (checkBoxExpirationDate.Checked)

bk.ExpirationDate = monthCalendar1.SelectionStart;//ssms default language en

string path = string.Format($"{databaseName}\_" +

$"{DateTime.Now.ToString("dd.MM.yyyy\_HH-mm-ss")}" +

$".bak");

BackupDeviceItem bdi = new BackupDeviceItem(path, DeviceType.File);

DateTime dt = DateTime.Now;

bk.Devices.Add(bdi);

bk.SqlBackup(dtb.srv);

bk.Devices.Remove(bdi);

}

private void RestorePercent\_Completed(object sender, PercentCompleteEventArgs e)

{

progressBar1.Invoke((MethodInvoker)delegate

{

progressBar1.Value = e.Percent;

progressBar1.Update();

});

label5.Invoke((MethodInvoker)delegate

{

label5.Text = $"{e.Percent}%";

});

progressBar1.Value = 0;

progressBar1.Update();

label5.Text = "0%";

}

void Restore\_Completed(object sender, ServerMessageEventArgs e)

{

if (e.Error != null)

{

MessageBox.Show(e.Error.Message);

}

}

private void buttonRecovery\_Click(object sender, EventArgs e)

{

OpenFileDialog OPF = new OpenFileDialog();

if (OPF.ShowDialog() == DialogResult.OK)

{

string filePath = OPF.FileName;

string databaseName = textBoxNewDatabase.Text;

Restore restore = new Restore();

restore.Action = RestoreActionType.Database;

restore.NoRecovery = false;

BackupDeviceItem bdi = new BackupDeviceItem(filePath, DeviceType.File);

Database currentDb = null;

restore.Complete += Restore\_Completed;

restore.PercentComplete += RestorePercent\_Completed;

restore.ReplaceDatabase = true;

if (radioButtonToOldDatabase.Checked)

{

if (comboBoxdataBasesforRestore.Text != null)

{

string db = comboBoxdataBasesforRestore.Text;

currentDb = dtb.srv.Databases[db];

if (currentDb != null)

{

dtb.srv.KillAllProcesses(currentDb.Name);

}

restore.Database = db;

restore.Devices.Add(bdi);

restore.SqlRestore(dtb.srv);

addAllDatabasesToLists();

}

}

else

{

Database newDb = new Database(dtb.srv, textBoxNewDatabase.Text);

restore.Database = newDb.Name;

restore.Devices.Add(bdi);

restore.SqlRestoreAsync(dtb.srv);

addAllDatabasesToLists();

}

}

}

private void comboBoxdataBasesforBackup\_SelectedIndexChanged(object sender, EventArgs e)

{

textBoxNameBase.Text = comboBoxdataBasesforBackup.SelectedItem.ToString();

}

private void radioButtonToOldDatabase\_CheckedChanged(object sender, EventArgs e)

{

radioButtonToNewDatabase.Checked = !radioButtonToOldDatabase.Checked;

}

private void radioButtonToNewDatabase\_CheckedChanged(object sender, EventArgs e)

{

radioButtonToOldDatabase.Checked = !radioButtonToNewDatabase.Checked;

}

private void comboBoxAuthentication\_SelectedIndexChanged(object sender, EventArgs e)

{

if (comboBoxAuthentication.SelectedIndex == 0)

textBoxNameUser.Enabled = textBoxPass.Enabled = false;

else textBoxNameUser.Enabled = textBoxPass.Enabled = true;

}

public void RefreshJobs()

{

dataGridView1.Rows.Clear();

foreach (Job item in dtb.srv.JobServer.Jobs)

dataGridView1.Rows.Add(item.JobID, item.Name, item.IsEnabled, item.DateCreated);

}

private void buttonNewJob\_Click(object sender, EventArgs e)

{

form2.ShowDialog();

RefreshJobs();

}

private void buttonDeleteJob\_Click(object sender, EventArgs e)

{

string nameJob = (string)dataGridView1.CurrentRow.Cells["Column2"].Value;

Job job = dtb.srv.JobServer.Jobs[nameJob];

job.Drop();

RefreshJobs();

}

private void buttonStartJob\_Click(object sender, EventArgs e)

{

string nameJob = (string)dataGridView1.CurrentRow.Cells["Column2"].Value;

Job job = dtb.srv.JobServer.Jobs[nameJob];

job.IsEnabled = true;

job.Alter();

job.Refresh();

RefreshJobs();

}

private void buttonStopJob\_Click(object sender, EventArgs e)

{

string nameJob = (string)dataGridView1.CurrentRow.Cells["Column2"].Value;

Job job = dtb.srv.JobServer.Jobs[nameJob];

job.IsEnabled = false;

job.Alter();

job.Refresh();

RefreshJobs();

}

//для ручного бекапа

//public async Task SendMessage(string toEmail, string title, string message)

//{

// MimeMessage myMessage = new MimeMessage();

// myMessage.From.Add(new MailboxAddress("testNameFrom","polyakovmymail@gmail.com"));

// myMessage.To.Add(new MailboxAddress("testNameTo",toEmail));

// myMessage.Subject = title;

// myMessage.Body = new TextPart(MimeKit.Text.TextFormat.Html)

// {

// Text = message

// };

// using (SmtpClient mySmtpClient=new SmtpClient())

// {

// await mySmtpClient.ConnectAsync("smtp.gmail.com", 465, false);

// await mySmtpClient.AuthenticateAsync("polyakovmymail@gmail.com", "passValue");

// await mySmtpClient.SendAsync(myMessage);

// await mySmtpClient.DisconnectAsync(true);

// }

//}

}

}

# Приложение 2

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using Microsoft.SqlServer.Management.Smo;

using Microsoft.SqlServer.Management.Smo.Agent;

using Microsoft.SqlServer.Management.Smo.Mail;

namespace databaseBackup

{

public partial class Form2 : Form

{

JobSchedule jbs;

DataBase dtb = DataBase.getInstance();

public Form2()

{

InitializeComponent();

}

private void Form2\_Load(object sender, EventArgs e)

{

comboBoxDataBases.Items.Clear();

foreach (Database item in dtb.srv.Databases)

{

comboBoxDataBases.Items.Add(item.Name);

}

}

private void buttonCreateJob\_Click(object sender, EventArgs e)

{

if (comboBoxDataBases.SelectedItem == null)

{

MessageBox.Show("Нужно выбрать базу данных");

return;

}

string email, profile;

if (textBoxEmail.Text == null)

{

email = "polyakovmymail@gmail.com";

}

else email = textBoxEmail.Text;

if (textBoxProfile.Text == null)

{

profile = "polyakovmymail@gmail.com";

}

else profile = textBoxProfile.Text;

Job job = new Job(dtb.srv.JobServer, JobName.Text);

////mailAcc example for send

//SqlMail sm = dtb.srv.Mail;

//MailAccount ma = default(MailAccount);

//Random rand = new Random();

//string mailAccountUniqueName = "UniqueName" + rand.Next() % 10000;

//ma = new MailAccount(sm, mailAccountUniqueName, "AdventureWorks2012 Automated Mailer",

// "Mail account for administrative e-mail.", "dba@Adventure-Works.com");

//ma.Create();

//string alertName,operatorName,operatorMail; //"textName"

//if (textBoxNotificationName.Text == null)

// alertName = "myalertName";

//else alertName = textBoxNotificationName.Text;

//if (textBoxNameOperator.Text == null)

// operatorName = "textName";

//else operatorName = textBoxNameOperator.Text;

//if (textBoxEmail.Text == null)

// operatorMail = "polyakovmymail@gmail.com";

//else operatorMail = textBoxEmail.Text;

//job.OperatorToEmail = operatorMail;

job.EmailLevel = CompletionAction.Always;

//Alert myAlert = new Alert(dtb.srv.JobServer, alertName);

//myAlert.Severity = 19;

//myAlert.NotificationMessage = "test";

//myAlert.Create();

//Operator myOperator = new Operator(dtb.srv.JobServer, textBoxNameOperator.Text);

//myOperator.EmailAddress = operatorMail;

//myOperator.Enabled = true;

//myOperator.Create();

//myAlert.AddNotification(operatorName, NotifyMethods.NotifyEmail);

//myOperator.AddNotification(alertName, NotifyMethods.NotifyEmail);

job.IsEnabled = true;

job.Create();

job.ApplyToTargetServer(dtb.serverName);

JobStep jobStep = new JobStep(job, "auto backups");

jobStep.SubSystem = AgentSubSystem.PowerShell;

jobStep.DatabaseName = "dataBaseTest";

jobStep.OnSuccessAction = StepCompletionAction.GoToNextStep;

jobStep.OnFailAction = StepCompletionAction.QuitWithFailure;

//директория бекапов по умолчанию

string backupDirectory = dtb.srv.BackupDirectory;

//полный бекап

string finalCommand = dtb.FullBackup(comboBoxDataBases.Text, backupDirectory);

finalCommand += dtb.EmailAndFreeSpace(email, profile);//add query email

jobStep.Command = finalCommand;

jobStep.Create();

jbs = new JobSchedule(job, JobName.Text);

jbs.ActiveStartDate = startDateTimeJob.Value;

if (radioButtonDaily.Checked)

{

jbs.FrequencyTypes = FrequencyTypes.Daily;

jbs.FrequencySubDayTypes = FrequencySubDayTypes.Once;

jbs.FrequencyInterval = 1;

jbs.ActiveStartTimeOfDay = new TimeSpan(daily.Value.Hour, daily.Value.Minute, daily.Value.Second);

}

jbs.Create();

job.AddSharedSchedule(jbs.ID);

}

}

}

# Приложение 3

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Microsoft.SqlServer.Management.Smo;

using Microsoft.SqlServer.Management.Smo.Agent;

using Microsoft.SqlServer.Management.Common;

namespace databaseBackup

{

class DataBase

{

private static DataBase instance;

private DataBase() { }

public static DataBase getInstance()

{

if (instance == null)

instance = new DataBase();

return instance;

}

public ServerConnection srvConn;

public Server srv;

public string serverName;

public string FullBackup(string NameBase, string Directory)

{

//Создание строки

string fileName = string.Format("{0}\_{1}.bak", NameBase, DateTime.Now.ToString("dd.MM.yyyy\_HH-mm-ss"));

string fullPath = Directory + "\\" + fileName;

string sqlExpression = $"BACKUP DATABASE [{NameBase}] " +

$"TO DISK = N'{fullPath}' " +

$"WITH FORMAT, NAME = N'{fileName} Database Full Backup'"; //Проверка на наличие объекта

return sqlExpression;

}

public string EmailAndFreeSpace(string email, string profile)

{

string sqlExpression= $"USE msdb"+

$"GO"+

$"EXEC sp\_send\_dbmail @profile\_name = '{profile}',"+

$"@recipients = '{email}',"+

$"@subject = 'Test message',"+

$"@body = 'someText',"+

$"@query = 'SELECT distinct(volume\_mount\_point)," +

$"total\_bytes / 1048576 as Size\_in\_MB," +

$"available\_bytes / 1048576 as Free\_in\_MB," +

$"(select((available\_bytes / 1048576 \* 1.0) / (total\_bytes / 1048576 \* 1.0) \* 100)) as FreePercentage" +

$"FROM sys.master\_files AS f CROSS APPLY" +

$"sys.dm\_os\_volume\_stats(f.database\_id, f.file\_id)" +

$"group by volume\_mount\_point, total\_bytes / 1048576," +

$"available\_bytes / 1048576 order by 1'," +

$"@attach\_query\_result\_as\_file = 1";

return sqlExpression;

//USE msdb

//GO

//EXEC sp\_send\_dbmail @profile\_name = 'finalTest',

//@recipients = 'augustxeno@gmail.com',

//@subject = 'Test message',

//@body = 'xp\_fixeddrives',

//@query = 'SELECT distinct(volume\_mount\_point),

// total\_bytes / 1048576 as Size\_in\_MB,

// available\_bytes / 1048576 as Free\_in\_MB,

// (select((available\_bytes / 1048576 \* 1.0) / (total\_bytes / 1048576 \* 1.0) \* 100)) as FreePercentage

//FROM sys.master\_files AS f CROSS APPLY

// sys.dm\_os\_volume\_stats(f.database\_id, f.file\_id)

//group by volume\_mount\_point, total\_bytes / 1048576,

// available\_bytes / 1048576 order by 1',

//@attach\_query\_result\_as\_file = 1;

}

}

}