УНИВЕРСИТЕТ ИТМО

И.С. Осетрова

Администрирование MS SQL Server 2014

Санкт-Петербург
2016

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

И.С. Осетрова

Администрирование MS SQL Server 2014

Учебное пособие

Санкт-Петербург

2016

УДК 004.655, 004.657, 004.62

И.С. Осетрова

Администрирование MS SQL Server 2014 - СПб: Университет ИТМО, 2016. – 90 с.

В пособии представлено руководство по основным приемам работы в MS SQL Server 2014 по дисциплине “ Создание клиент-серверных приложений”.

Предназначено для студентов, обучающихся по образовательной программе: 11.03.02 «Интеллектуальные инфокоммуникационные системы» (бакалавр).

Рекомендовано к печати Ученым советом факультета инфокоммуникационных технологий, протокол № 09/16 от 24.11.2016г

Университет ИТМО – ведущий вуз России в области информационных и фотонных технологий, один из немногих российских вузов, получивших в 2009 году статус национального исследовательского университета. С 2013 года Университет ИТМО – участник программы повышения конкурентоспособности российских университетов среди ведущих мировых научно-образовательных центров, известной как проект «5 в 100». Цель Университета ИТМО – становление исследовательского университета мирового уровня, предпринимательского по типу, ориентированного на интернационализацию всех направлений деятельности.

ОГЛАВЛЕНИЕ

Введение

Учебное пособие предназначено для студентов, обучающихся по профилю подготовки бакалавров направления 11.03.02 «Интеллектуальные инфокоммуникационные системы».

В курсе «Создание клиент-серверных приложений» студенты должны овладеть навыками разработки и создания баз данных, отвечающих всем требованиям согласованности, безопасности и производительности. Это, в свою очередь, требует знаний не только в области собственно разработки, но и знание средств, позволяющих в процессе эксплуатации баз данных проводить их оперативное обслуживание и сопровождение.

Основная цель учебного пособия состоит в ознакомлении со средствами администрирования MS SQL Server 2014, которые предназначены для решения вопросов разработки и создания производительных реляционных баз данных, а также их дальнейшего обслуживания и анализа.

Дополнительные сведения можно найти в списке рекомендуемой литературы, в первую очередь в электронной документации Microsoft [1-3].

1 ВВЕДЕНИЕ В АДМИНИСТРИРОВАНИЕ БАЗ ДАННЫХ SQL Server

Большинство организаций используют приложения для управления бизнес-процессами и действиями, а эти приложения обычно хранят данные в базе данных (БД). Организации все более и более зависят от приложений и данных, которые они хранят. В такой ситуации базы данных являются критически важным компонентом бизнеса в области инфраструктуры информационных технологий.

Роль администратора базы данных (database administrator, DBA) включает широкий круг обязанностей и задач, которые гарантируют, что эти базы данных оптимально хранятся, постоянно поддерживаются в согласованном состоянии и используются с высокой производительностью.

1.1 Обязанности и задачи DBA

Основные требования, предъявляемые к администраторам баз данных

- **Технологические знания и навыки**. Администрирование баз данных требует не только глубокого знания платформы, используемой для размещения баз данных, но также знаний в области конфигураций операционной системы, устройств хранения данных и сетей.
- **Бизнес-осведомленность.** Администратор баз данных должен понимать бизнес-контекст, в котором функционирует база данных, и ее роль в поддержке бизнеса.
- **Организационные навыки.** Системы баз данных могут быть сложными, с большим количеством компонентов и подсистем. Некоторые задачи должны выполняться в определенное время. Хороший администратор должен отслеживать эти задачи, а также оперативно реагировать на неожиданные проблемы в случае их возникновения.
- **Умение выстраивать приоритеты**. Когда возникают неожиданные проблемы, которые могут негативно повлиять на работу с базой данных, администратор должен грамотно расположить их решение по приоритетам, основываясь на таких факторах, как соглашения об уровне обслуживания (service level agreement, SLA), число пользователей и затронутых систем, а также степень влияния возникшей проблемы на текущие операции.

Общие задачи администрирования баз данных

- **Подготовка баз данных и серверов баз данных**. Это может включать установку и настройку экземпляров SQL Server на физических или виртуальных серверах или создание новых виртуальных машин на основе шаблонов изображений, а также создание баз данных и распределение их данных и файлов журналов на устройствах хранения.
- **Сохранение файлов баз данных и объектов**. После того, как база данных создана и заполняется данными, для оптимальной работы требуется ее постоянное обслуживание и оптимизация. Это предполагает

уменьшение фрагментации, которая появляется по мере того, как записи добавляются и удаляются, сохранение файлов данных соответствующего размера и обеспечение последовательной структуры логических и физических данных.

- **Управление восстановления в случае сбоя базы данных**. Базы данных часто имеют решающее значение для деловых операций, поэтому главной задачей DBA является планирование соответствующей стратегии резервного копирования и восстановления для каждой базы данных, что позволило бы осуществить восстановление базы данных в случае сбоя.
- **Импорт и экспорт данных**. Данные часто передаются между системами, поэтому администраторам баз данных необходимо выполнять экспорт или импорт данных.
- **Применение безопасности к данным**. Серверы баз данных организации содержат данные, которые позволяют бизнесу работать. Нарушение безопасности может быть дорогостоящим и трудоемким для восстановления, приводить к потере доверия клиентов. Администратор баз данных должен реализовывать такие политики безопасности, которые обеспечивают пользователям доступ к необходимым данным, но при этом соблюдают правовые нормы бизнеса по защите своих активов, а также снижают риски, связанные с нарушением безопасности.
- **Мониторинг и устранение неполадок систем баз данных**. Многие операции по администрированию баз данных являются реактивными, то есть они предполагают принятие мер для устранения неполадок и возникающих проблем. Грамотные администраторы БД осуществляют упреждающий подход, чтобы попытаться обнаружить потенциальные проблемы до того, как они начнут влиять на операции с данными.

Документирование процедур управления базой данных

Большинство администраторов баз данных знакомы с системами, которыми они управляют, и знают задачи, которые должны выполняться ежедневно. Однако даже опытные DBA не полагаются исключительно на свою память. Администраторы БД обычно составляют и ведут документацию (“run book”), которая включает в себя такие сведения, как:

- Параметры конфигурации и расположения файлов,
- Контактная информация персонала,
- Стандартные правила и графики технического обслуживания,
- Процедуры аварийного восстановления.

Ведение документации является важной частью администрирования баз данных. Подробная книга может иметь неоценимое значение, особенно в случае, когда новый администратор должен взять на себя ответственность за управление базой данных, или при возникновении неожиданной чрезвычайной ситуации в отсутствии администратора. При сбое сервера четко задокументированные шаги восстановления базы данных уменьшают чувство паники и обеспечивают быстрое решение проблемы.

1.2 Обзор платформы SQL Server

Очень важно хорошо знать систему управления базами данных (СУБД), которая используется для хранения данных. СУБД SQL Server является платформой, ориентированной на разработку бизнес-приложений.

Выпуски SQL Server

SQL Server поставляется в различных выпусках, перечень которых приведен в Таблице 1. Важно правильно выбрать выпуск SQL Server, а для этого надо понимать их назначение и возможности [1-3].

Таблица 1. Выпуски SQL Server 2014

Premium			
	Parallel Data Warehouse		Enterprise
Базовые			
	Business Intelligence		Standard
Другие			
	Express	Compact	Developer
	Web	Microsoft Azure SQL Database	

Основные выпуски:

- **Enterprise**, который является ведущим выпуском. Содержит все функции SQL-сервера, включая службы BI и поддержку виртуализации.
- **Standard**, включающий базовый механизм database engine, а также базовое создание отчетов и возможности аналитики. Однако поддерживает меньше ядер процессора и не предлагает всех возможностей, безопасности и функций организации хранилищ данных, представленных в версии Enterprise.

Компоненты SQL Server

Важно понимать, что SQL Server не является единым монолитным приложением, а структурирован как ряд компонентов. Компоненты SQL Server 2014 и их назначение приведены в Таблице 2.

Таблица 2. Компоненты SQL Server 2014

Компонент	Описание
Database Engine	Ядро («сердце») платформы SQL Server. Обеспечивает высокую производительность и масштабируемость реляционных баз данных на основе языка SQL. Может использоваться для размещения данных, для их дальнейшей обработки с помощью транзакций в интерактивном режиме (OLTP - Online Transaction Processing) и для создания хранилищ данных (data warehouse). SQL Server 2014 также включает в себя оптимизированный движок базы данных, который использует технологию «in-memory» для повышения производительности коротких транзакций.

Компонент	Описание
Analysis Services	SQL Server Analysis Services (SSAS) – аналитические службы – обеспечивают функциональность OLAP (Online Analytical Processing) и анализа данных для приложений бизнес-аналитики (data mining). Они позволяют организации собирать данные из нескольких источников, например, реляционных БД, и обрабатывать их различными способами.
Integration Services	SQL Server Integration Services (SSIS) – службы интеграции – инструмент масштаба предприятия для извлечения, преобразования и интеграции данных (ETL –extract, transform, and load) из различных источников и их перемещения в один или несколько целевых источников данных. Предназначены для слияния данных из разнородных источников и загрузки их в хранилища, витрины данных и пр.
Reporting Services	SQL Server Reporting Services (SSRS) – службы отчетов – включают Диспетчер отчетов (Report Manager) и Сервер отчетов (Report Server). Представляют собой полномасштабную серверную платформу для создания, управления и распространения отчетов. Позволяют использовать для обработки и хранения отчетов сочетание возможностей SQL Server и IIS. SSRS может быть установлен самостоятельно или интегрирован с Microsoft SharePoint Server
Master Data Services	SQL Server Master Data Services (MDS) среда для создания бизнес-правил, гарантирующих качество и точность основных данных. Бизнес-правила могут применяться для запуска бизнес-процессов, выполняющих проверки и управляющих потоками данных.
Data Quality Services	SQL Server Data Quality Services (DQS) среда для создания хранилища метаданных базы знаний, что позволяет улучшить качество данных организации. Процессы очистки данных позволяют изменять или удалять неполные и некорректные данные, процессы сопоставления позволяют выявлять и объединять дублирующиеся данные.
StreamInsight	SQL Server StreamInsight предоставляет платформу для создания приложений, которые выполняют обработку сложных событий для потоков данных в реальном времени.
Full-Text Search	Полнотекстовый поиск – это функция компонента database engine, который обеспечивает механизм сложного семантического поиска для текстовых данных.
Replication	Database Engine включает в себя репликации, набор технологий для синхронизации данных между серверами для удовлетворения потребностей распространения данных
Power View for SharePoint Server	Power View – компонент SQL Server Reporting Services (при установке в режиме SharePoint-Integrated mode). Это обеспечивает интерактивное исследование данных, их визуализацию и презентацию. Power View также доступен в Excel.

Основными компонентами являются Database Engine, SSAS, SSIS, SSRS.

Экземпляры SQL Server

Многие компоненты SQL Server можно установить более чем один раз в виде отдельных экземпляров (*instance*) сервера. Каждый экземпляр может настраиваться и управляться независимо.

В следующих ситуациях полезно установить больше чем одну копию компонента SQL сервера на одном сервере:

- Можно управлять и защитить каждый экземпляр SQL Server отдельно. Поэтому можно для разных наборов баз данных иметь различных администраторов и/или различные среды безопасности.
- Каждый экземпляр SQL Server можно настроить самостоятельно (независимо друг от друга). Некоторым из приложений может потребоваться конфигурация сервера, которая не соответствует или несовместима с требованиями других приложений.
- Экземпляры SQL Server можно использовать для разделения рабочих нагрузок с различными соглашениями об уровне обслуживания (SLAs). Приложениям базы данных могут потребоваться различные уровни обслуживания, особенно в отношении доступности.
- Может потребоваться поддержка различных версий и выпусков SQL Server.
- Приложениям могут требоваться различные параметры сортировки на уровне сервера. Хотя каждая база данных может иметь разные параметры сортировки, приложение может зависеть от параметров сортировки базы данных tempdb, когда использует временные объекты.

Различные версии SQL Server также могут быть установлены с помощью нескольких экземпляров. Это может помочь при тестировании сценариев обновления или выполнения обновлений.

1.3 Инструменты и методы управления базами данных

SQL Server имеет целый ряд инструментов для управления различными аспектами СУБД.

SQL Server Management Studio

SQL Server Management Studio (SSMS) является основным инструментом управления базами данных для серверов баз данных SQL Server. Он представляет собой графический пользовательский интерфейс (GUI) и интерфейс сценариев Transact-SQL для управления компонентом ядра базы данных и базами данных. Кроме того, можно использовать SSMS, чтобы управлять экземплярами SSAS, SSIS и SSRS, а также базами данных на базе облака в Microsoft Azure SQL Database. Вид окна SSMS представлен на рисунке 1.

SQL Server Configuration Manager

SQL Server Configuration Manager (SSCM) можно использовать для настройки и управления службами SQL Server, а также для управления протоколами сети клиента и псевдонимами.

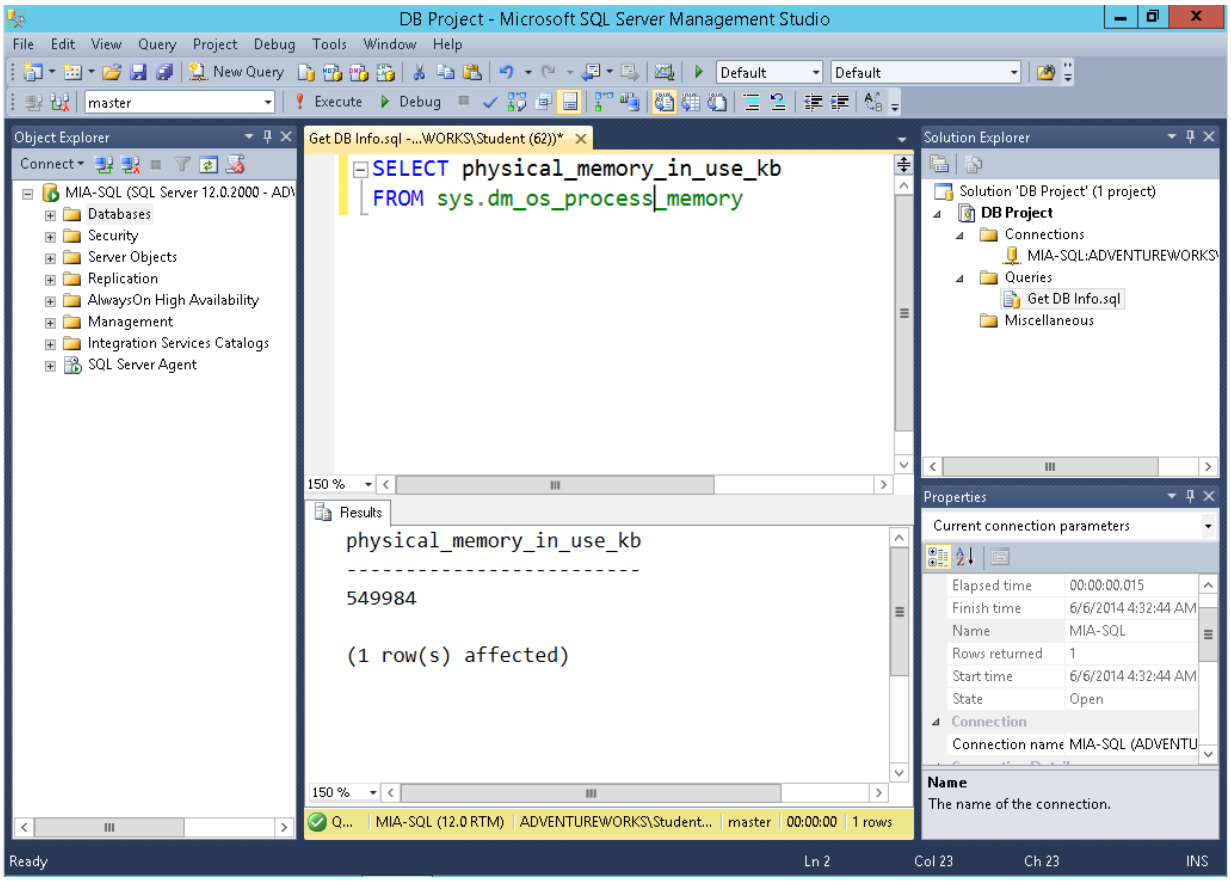

Рисунок 1 Интерфейс SQL Server Management Studio

SQL Profiler

Можно использовать SQL Profiler в случаях, когда необходимо изучить работу базы данных SQL Server или для записи трассировки. Это позволяет исследовать и устранять проблемы, а также оптимизировать конфигурацию базы данных на основе фактических моделей использования. Однако в настоящее время этот инструмент является устаревшим и заменен расширенными событиями.

SQL Server Database Engine Tuning Advisor

Помощник по настройке ядра СУБД SQL Server (DTA). Правильно оптимизированная база данных использует индексы и другие структуры для повышения производительности запросов. DTA дает рекомендации на основе анализа типичных рабочих нагрузок базы данных и может служить полезной отправной точкой для оптимизации баз данных.

SQL Server Import and Export

Этот инструмент представляет собой графический мастер, который упрощает процесс передачи данных в базу данных или из базы данных.

Sqlcmd utility

Произносится "SQL Command". Это инструмент командной строки, который можно использовать для подключения к экземпляру SQL Server и запускать инструкции Transact-SQL или сценарии (см. рисунок 2).

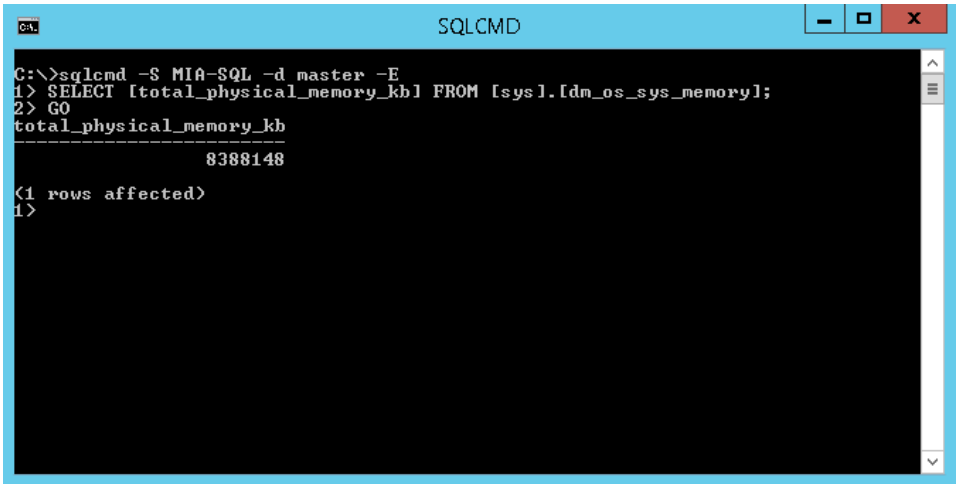

Рисунок 2 Вид инструмента sqlcmd utility

Sqlcmd используют для автоматизации задач базы данных из командной строки, а также для выполнения задач настройки и управления, когда среда SSMS недоступна. В частности, программа sqlcmd можно использовать для открытия выделенного административного соединения (DAC) на сервер когда стандартные соединения невозможны.

Программа sqlcmd предоставляет параметры, которые можно использовать для настройки подключений и выполнения задач. Эти параметры включают:

- S имя_сервера (подключение к указанному серверу)
- d имя_базы_данных (подключиться к указанной базе данных)
- U login (вход в качестве указанного имени входа)
- P пароль (подлинность имени входа с указанным паролем)
- E (используйте доверенное соединение для проверки подлинности Windows)
- A (Открыть выделенное административное соединение)
- i входной_файл (запустить код Transact-SQL в указанном входном файле)
- o выходной_файл (сохранить выходные данные в указанный файл)
- q «Запрос Transact-SQL» (выполнить указанный запрос)
- Q «Запрос Transact-SQL» (выполнить указанный запрос и выход)
- var v = «value» (передайте указанную переменную к входной скрипт)

Программа sqlcmd поддерживает многие другие параметры. Для получения полного списка в командной строке введите «sqlcmd-?».

Bcp utility

BCP (Bulk Copy Program) означает программы массового копирования, и утилита bcp является инструментом командной строки для импорта и экспорта данных в SQL Server.

Использование Transact-SQL для выполнения задач управления

Можно выполнять большинство административных задач в среде SSMS с помощью графического пользовательского интерфейса. Однако некоторые задачи могут быть выполнены только с помощью инструкций Transact-SQL; и даже если задача может быть выполнена в графическом интерфейсе, целесообразно использовать код Transact-SQL, который может быть сохранен в виде сценария и повторно выполняться (или запускаться автоматически с помощью планировщика).

Большинство графических интерфейсов в SSMS имеют кнопку Script, с помощью которой автоматически генерируется эквивалентный код Transact-SQL.

Команды Transact-SQL, которые можно использовать для выполнения задач управления, включают:

- **Инструкции языка DDL**. Например, можно использовать инструкции «CREATE DATABASE» или «DROP DATABASE» для создания БД или для удаления базы данных.
- **Системные хранимые процедуры и функции**. SQL Server предоставляет системные хранимые процедуры и функции, которые инкапсулируют общие задачи настройки и управления системы. Например, можно использовать системную хранимую процедуру sp_configure для задания параметров конфигурации экземпляра SQL Server.
- **DBCC (Database Console Commands)**. Команды DBCC используются для выполнения конкретных задач по настройке и обслуживанию, а также для выполнения проверок в базе данных. Например, можно использовать команду DBCC CHECKDB для проверки физической и логической целостности объектов в базе данных.

2 РАБОТА С БАЗАМИ ДАННЫХ

Одной из наиболее важных задач администратора баз данных, работающих с Microsoft SQL Server, является управление базами данных и хранением данных. Поэтому важно знать, как создавать базы данных, как данные в базах данных хранятся, как управлять файлами базы данных. Есть и другие задачи, связанные с хранением, например, управление базой данных tempdb.

2.1 Хранение данных в SQL Server

Для успешного выполнения задач администрирования баз данных надо знать, как данные хранятся в SQL сервере.

Логически базы данных SQL Server состоят из таблиц и других объектов. Физически эти данные представляют собой набор файлов.

Существует три типа файлов базы данных SQL Server: первичный файл данных, вторичные файлы данных и файлы журналов транзакций (см. рисунок 3).

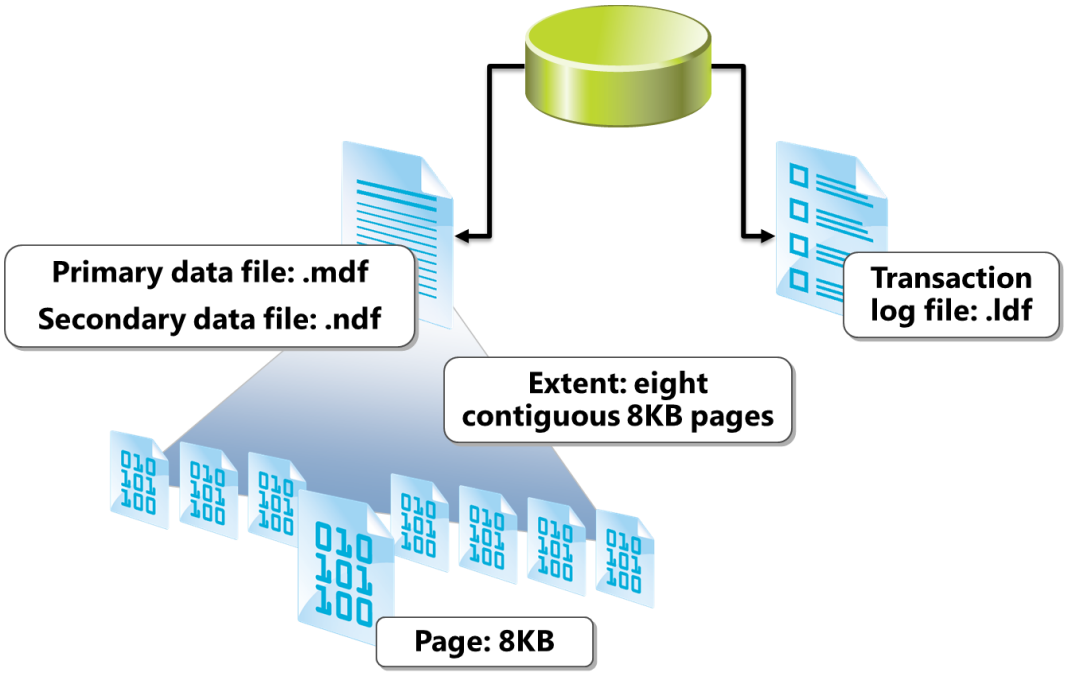

Рисунок 3 Физическая структура базы данных

Первичный файл данных является отправной точкой базы данных. В каждой базе данных обязательно имеется один первичный файл данных. Кроме собственно страниц данных, первичный файл содержит указатели на другие файлы базы. Первичные файлы данных обычно используют расширение **.mdf**. Использование этого расширение файла не является обязательным, но настоятельно рекомендуется.

Вторичные файлы данных являются необязательными, определяются пользователем как дополнительные, которые могут быть использованы для хранения данных в нескольких местах (для повышения производительности и/или удобства технического обслуживания). Вторичные файлы можно

использовать для распределения данных по нескольким дискам, поставив каждый файл на отдельный диск. Кроме того, если размер базы данных превышает максимальный размер одного файла Windows, можно использовать вторичные файлы данных, чтобы база данных могла продолжать расти. Рекомендуемое расширение для вторичных файлов данных – **.ndf**.

Файлы журналов транзакций (или файлы журнала) содержат сведения, которые при необходимости можно использовать для восстановления базы данных. Должен быть хотя бы один файл журнала для каждой базы данных. Все транзакции записываются в файл журнала, используя механизм «упреждения» (WAL – write-ahead logging) для обеспечения целостности базы данных в случае сбоя и для поддержки отката транзакций. Рекомендуемое расширение для файлов журнала – **.ldf**.

Файл журнала также используется в других компонентах SQL Server, таких как репликация транзакций, зеркальное отображение базы данных и сбор изменения данных.

Страницы и экстенты файлов данных. Файлы данных хранят данные в виде страниц, которые сгруппированы в экстенты. Дисковые операции ввода-вывода выполняются на уровне страницы. Каждая страница имеет размер 8 КБ. Страница имеет служебную область – заголовок (96 байт), собственно для хранения данных остается область 8 096 байт. Страницы в файле данных нумеруются последовательно, начиная с нуля. Каждый файл в базе данных также имеет свой уникальный идентификационный номер. Для уникальной идентификации страницы в базе данных требуется идентификатор файла и номер страницы. Строки данных могут содержать значения столбцов фиксированной и переменной длины. Для одной записи все столбцы фиксированной длины должны поместиться на одной странице в 8 060 байт. Данные таблицы и индекса хранятся только в страницах.

Группы из восьми непрерывных страниц называются экстентами. SQL Server использует экстенты для упрощения управления памятью в файлах данных. Существует два типа областей:

- *Однородные экстенты* принадлежат одному объекту: все восемь страниц могут быть использованы только этим владеющим объектом;
- *Смешанные экстенты* могут находиться в общем пользовании у не более восьми объектов: каждая из восьми страниц экстента может содержать данные из разных объектов.

В первичных и вторичных файлах данных выделяется небольшое количество страниц для отслеживания использования экстентов в файле.

Уровни RAID. Многие решения для хранения данных используют аппаратное обеспечение RAID для реализации отказоустойчивости и повышения производительности. Контролируемое программное обеспечение RAID 0, RAID 1, RAID 5 также можно реализовать с помощью операционной системы Windows Server.

Часто используемые типы RAID показаны на рисунке 4:

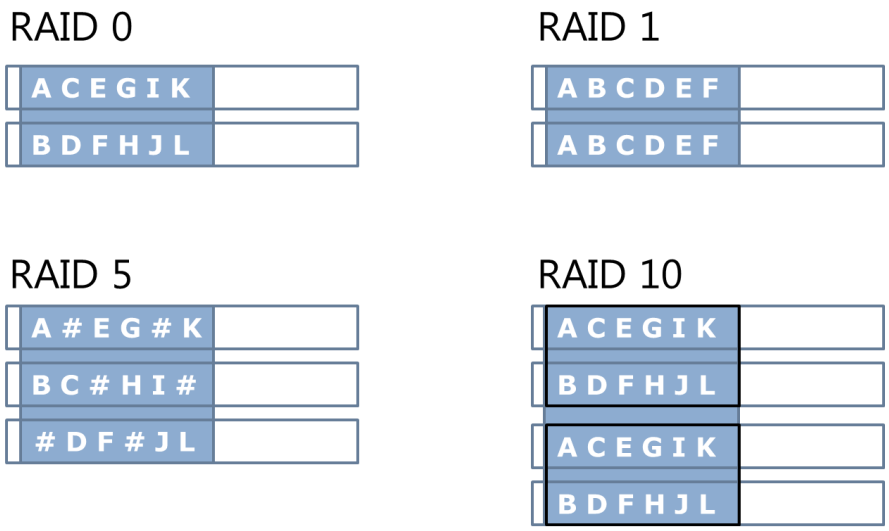

Рисунок 4 Уровни RAID

- **RAID 0** (чередование дисков). Чередующийся набор состоит из двух или более дисков, объединенных в один том. Данные равномерно распределяются по всем дискам, что повышает производительность операций ввода-вывода; особенно когда каждый диск имеет свой собственный аппаратный контроллер.
- **RAID 1** (зеркальное отображение дисков). Зеркальное отображение дисков означает наличие точной дополнительной копии выбранного диска. Все данные, записываемые на основной диск, также записываются и на зеркальный диск. Уровень RAID 1 обеспечивает отказоустойчивость и, в целом, улучшает производительность считывания, но может снизить производительность записи.
- **RAID 5** (чередование с четностью). RAID 5 обеспечивает высокую отказоустойчивость за счет использования данных четности, которые записываются на все диски чередующегося тома, который состоит из 3 или более дисков. Чередование с четностью обеспечивает лучшую производительность, чем RAID 1. Однако в случае сбоя диска в наборе производительность ухудшается. RAID 5 является менее дорогостоящим с точки зрения дискового пространства, чем RAID 1.
- **RAID 10** (зеркальное отображение с чередованием). Иногда говорят, что RAID 10 – это (RAID 1 + RAID 0). Этот механизм обеспечивает отличную производительность чтения/записи RAID 0 в сочетании с отказоустойчивости RAID 1. Однако дисков используется в два раза больше.

При планировании хранения файлов на оборудовании RAID надо учитывать, что RAID 10 предлагает наилучшее сочетание производительности чтения/записи и отказоустойчивости, но является самым дорогостоящим решением. Запись операций на RAID 5 иногда может быть довольно медленным по сравнению с RAID 1 из-за необходимости вычисления данных четности.

При создании базы данных необходимо решить, где хранить файлы базы данных. Выбор места хранения является чрезвычайно важным, так как в процессе эксплуатации это может иметь значительное влияние на производительность, отказоустойчивость, возможности восстановления и управляемости базы данных.

Важно разнести по разным дискам файлы журналов и файлы данных по соображениям производительности и восстановления.

Во-первых, методы доступа к файлам журналов и к файлам данных очень разные. Доступ к данным в файлах журналов состоит в основном из последовательных, синхронных операций записи. Доступ к данным в файлах данных преимущественно асинхронный и случайный.

Во-вторых, если потерян файл данных, база данных может быть восстановлена из резервной копии и журнала транзакций до момента сбоя. Если потерян файл журнала SQL Server, база данных может принудительно восстановить из файлов данных с возможностью потери данных или несогласованности в базе данных. Однако если файлы данных и журналов на одной дисковой подсистеме, которая потеряна, варианты восстановления обычно включают восстановление базы данных из предыдущей резервной копии с потерей всех транзакций с того времени. Изоляция данных и файлов журналов может помочь избежать наихудших последствий сбоев диска.

Эта изоляция файлов данных и файлов журналов должна быть на уровне физического диска!

Для хранения данных используют логические тома в качестве единиц хранения, и общая ошибка заключается в том, чтобы поместить файлы данных и журналов на разных томах, которые фактически основаны на одном и том же физическом накопителях. При изоляции файлов данных и журналов, убедитесь, что тома, на которых хранятся файлы данных и файлы журнала, основаны на отдельных физических устройствах хранения данных.

В идеале, все файлы данных для одной базы данных должны быть одинакового размера. Данные равномерно распределяются по всем доступным файлам данных. Производительность в данном случае повышается, если файлы распределены по разным местам хранения.

Выделение нескольких файлов данных предоставляет ряд преимуществ управления, включая:

- Возможность перемещения файлов и части данных.
- Сокращение времени восстановления, например, если только часть данных повреждена.
- Увеличение параллельности операций ввода / вывода
- Возможность иметь базы данных больше, чем максимальный размер одного файла Windows.

2.2 Управление хранением баз данных

Системные базы данных

SQL Server использует системную базу данных для поддержания внутренних метаданных. Администраторы базы данных должны быть знакомыми с системными базами данных SQL сервера и уметь управлять ими.

Экземпляр SQL Server всегда содержит пять системных баз данных – ***master***, ***msdb***, ***model***, ***tempdb***, и ***resource***, описание которых приведено в Таблице 3.

Таблица 3. Системные базы данных SQL Server

System Database	Описание
master	Хранит все системные настройки. Все, что определяется на уровне экземпляра сервера, обычно хранится в базе данных master. Если база данных master повреждена, SQL Server не запускается, поэтому важно регулярно делать резервную копию БД master.
msdb	Содержит данные конфигурации агента SQL Server. База данных msdb содержит сведения о задачах обслуживания баз данных. Важно регулярно создавать резервные копии базы данных msdb, чтобы не потерять истории для резервного копирования, восстановления и планы обслуживания. В ранних версиях SQL Server пакеты служб SQL Server Integration Services (SSIS) часто хранились в базе данных msdb. В SQL Server 2014 следует хранить их в выделенной базе данных каталога служб SSIS.
model	Предоставляет шаблон для новых баз данных. Любая новая база данных использует базу данных model в качестве шаблона. Если создать объекты в базе данных model, они будут присутствовать во всех новых базах данных на экземпляре сервера. Обратите внимание, что SQL Server не запустится, если база данных model отсутствует.
tempdb	Содержит временные данные.
resource	Скрытая база данных только для чтения, содержит системные объекты, которые отображаются в схеме ***sys*** каждой базы данных. Эта база данных также содержит все системные хранимые процедуры, системные представления и системные функции. В версиях до SQL Server 2005 эти объекты были определены в базе данных master.

Все системные базы данных, за исключением базы данных resource, можно переместить на новые места, чтобы сбалансировать нагрузку ввода/вывода. Однако необходимо проводить перемещение системных баз данных с осторожностью, т. к. если это выполнить неправильно, то можно остановить работу SQL Server.

Перемещение файлов баз данных *msdb*, *model* и *tempdb* выполняется с помощью оператора `ALTER DATABASE … MODIFY FILE`.

Процесс перемещения базы данных *master* отличается. Перед перемещением файлов master.mdf и mastlog.ldf, надо открыть SQL Server Configuration Manager и в окне Свойств изменить значения параметров запуска для экземпляра SQL Server, чтобы указать планируемое место для файла данных (-d параметр) и файла журнала (-l параметр).

Производительность базы данных *tempdb* имеет решающее значение для общей производительности большинства установок SQL Server. Большинство объектов, которые находятся в базе данных *tempdb*, создаются пользователями и представляют собой временные таблицы, табличные переменные, результирующие наборы многоинструкционных функций и другие наборы временных строк.

По умолчанию файлы данных и журналов для базы данных *tempdb* хранятся в том же месте, что и файлы для всех других системных баз данных. Если экземпляр SQL Server должен поддерживать рабочие нагрузки баз данных, которые широко используют временные объекты, надо рассмотреть вопрос о переносе базы данных *tempdb* на выделенный том, чтобы избежать фрагментации файлов данных. Выбрать расположение файлов базы данных *tempdb* можно во время установки, а можно переместить их позже, если потребуется.

Поскольку база данных *tempdb* используется для многих целей, трудно заранее спрогнозировать необходимый размер. Следует тщательно протестировать и контролировать размер базы данных *tempdb* в реальных сценариях для новых установок. Недостаточно места на диске в базе данных *tempdb* может вызвать значительные нарушения в производственной среде SQL Server.

Создание и хранение пользовательских баз данных

Создавать пользовательские базы данных можно с помощью пользовательского графического интерфейса в среде SSMS или командой CREATE DATABASE. Инструкция CREATE DATABASE предлагает более гибкие варианты, но в использовании пользовательский интерфейс может быть проще.

Имена баз данных должны быть уникальными внутри экземпляра SQL Server и соответствовать правилам для идентификаторов.

Имя базы данных имеет тип sysname, который определяется как nvarchar(128). Это означает, что в имени базы данных могут присутствовать до 128 символов, и что каждый символ может быть выбран из набора символов Юникода (двухбайтовых). Однако с длинными именами баз данных неудобно работать.

В следующем примере кода T-SQL создается база данных с именем Sales, состоящая из двух файлов: первичный файл данных расположен в M:\Data\Sales.mdf, и файл журнала – в L:\Logs\Sales.ldf:

```
CREATE DATABASE Sales
ON
(NAME = Sales_dat,
FILENAME = 'M:\Data\Sales.mdf',
SIZE = 100MB, MAXSIZE = 500MB, FILEGROWTH = 20%)
LOG ON
(NAME = Sales_log,
FILENAME = 'L:\Logs\Sales.ldf',
SIZE = 20MB, MAXSIZE = UNLIMITED, FILEGROWTH = 10MB);
```

Для каждого файла указывается логическое имя, а также полный физический путь к файлу. Поскольку операции в SQL Server для ссылки на файл используют логическое имя файла, оно должно быть уникальным в пределах каждой базы данных.

В этом примере первичный файл данных имеет начальный размер 100 МБ и максимальный размер 500 МБ. Он будет расти на 20 процентов от его текущего размера всякий раз, когда должно произойти авторасширение. Файл журнала имеет исходный размер 20 МБ и не имеет ограничений на максимальный размер файла. Каждый раз, когда ему необходимо автоматическое расширение, он будет расти на фиксированную величину в 10 МБ.

Если параметр сортировки не указан, то он будет по умолчанию таким, как параметр сортировки, указанный для экземпляра сервера во время установки. При необходимости, конкретные параметры сортировки могут быть выделены на уровне базы данных.

Удалять пользовательские базы данных также можно с помощью пользовательского графического интерфейса в среде SSMS или инструкцией DROP DATABASE. Удаление базы данных автоматически удаляет все ее файлы.

В следующем примере база данных Sales удаляется:

```
DROP DATABASE Sales
```

Каждая база данных имеет набор параметров, которые можно настроить. Эти параметры являются уникальными для каждой базы данных, их изменение для одной базы данных не влияет на другие базы данных. Изначально все параметры базы данных настроены при ее создании из конфигурации базы данных *model*. Изменить значения параметров можно с помощью предложения SET оператора ALTER DATABASE или на странице свойств базы данных в SSMS.

Существует несколько категорий параметров базы данных:

- Параметры авто (Auto options). Контролируют некоторые автоматические модели поведения. Как правило, в большинстве систем рекомендуется отключить параметры *Auto Close* и *Auto Shrink*, но включить автоматическое создание и обновление статистики (*Auto Create Statistics* и *Auto Update Statistics*).
- Параметры курсора (Cursor options). Управление поведением курсора. Использование курсоров в работе не рекомендуется за исключением конкретных приложений, таких как утилиты. Следует отметить, что чрезмерное использование курсоров является распространенной причиной проблем с производительностью.
- Параметры доступности базы данных (Database availability options). База данных может находиться в оперативном или автономном режиме (*online* или *offline*), или она находится в режиме *read-only*.

- Параметры обслуживания и восстановления (Maintenance and recovery options). Модели восстановления базы данных рассматриваются в главе 4. В SQL Server 2005 была добавлена новая опция проверки контрольной суммы. Если параметр проверки страниц (*Page verify*) установлен в «CHECKSUM», для каждой страницы вычисляется и добавляется контрольная сумма, и всякий раз, когда страница извлекается с диска, происходит перепроверка контрольной суммы.

  Контрольная сумма добавляется к странице только после записи страницы. Включение опции не вызывает перезаписи страниц с контрольной суммой.

В процессе использования базы данных может потребоваться внести изменения, например, изменить имя или параметры. Такие изменения можно выполнить с помощью среды SSMS или с помощью инструкции языка T-SQL ALTER DATABASE:

```
ALTER DATABASE ... SET <опция>,
```

указав имя параметра и, где это применимо, значение для использования.

Многие параметры базы данных, которые настраиваются с помощью инструкции ALTER DATABASE, можно переопределить с помощью параметра задания уровня сеанса. Это позволяет пользователям или приложениям выполнять инструкции SET, чтобы настроить параметры только для текущей сессии.

Например, следующий код установит базу данных в режим *read-only*:

```
ALTER DATABASE HistoricSales
SET READ_ONLY;
```

Инструкция `ALTER DATABASE ... SET COMPATIBILITY_LEVEL...` установить совместимость базы данных с SQL Server более ранних версий. Значения параметра приведены в Таблице 4.

Таблица 4. Значения параметра COMPATIBILITY_LEVEL для SQL Server

Значение параметра	Версия сервера
80	SQL Server 2000
90	SQL Server 2005
100	SQL Server 2008 and SQL Server 2008 R2
110	SQL Server 2012
120	SQL Server 2014

Например, выполнение кода

```
ALTER DATABASE Sales
SET COMPATIBILITY_LEVEL = 100;
```

установит совместимость базы данных `Sales` с SQL Server 2008 / 2008 R2.

Дополнительные сведения можно найти в электронной документации по SQL Server [1-3].

Использование Файловых групп (Filegroups)

Базы данных состоят, по меньшей мере, из двух файлов: первичного файла данных и файла журнала транзакций. Для повышения производительности и управляемости больших баз данных, можно добавить вторичные файлы данных.

Файлы данных всегда определяются в пределах файловой группы! *Создание файловых групп упрощает управление физическим хранением файлов базы данных и позволяет хранить файлы с аналогичными требованиями вместе.*

Файловые группы представляют собой именованные коллекции файлов данных, которые можно использовать для упрощения размещения данных и выполнения задач администрирования, например резервного копирования и восстановления. Использование файловых групп также может повысить производительность базы данных, поскольку они позволяют распределить объекты базы данных (таблицы и индексы) на нескольких томах хранения.

Каждая база данных имеет первичную файловую группу (PRIMARY). При добавлении вторичных файлов данных в базу они автоматически попадают в первичную файловую группу, если при их создании не указана другая файловая группа (пользовательская).

Файл базы данных может принадлежать только к одной файловой группе. Файловая группа может использоваться только одной базой данных.

Файловые группы можно использовать для управления размещением данных. Например, если база данных содержит таблицы данных только для чтения, можно поместить эти таблицы в выделенной файловой группе, которая будет доступна только для чтения.

Кроме того, можно создавать разностные резервные копии, т.е. выполнять резервное копирование и восстановление файлов и файловых групп по отдельности. Это позволяет уменьшить время резервного копирования, потому что нужно копировать не всю базу данных, а только отдельные файлы или группы файлов, которые изменились со времени последнего резервного копирования.

Аналогичным образом можно достичь эффективности, когда речь заходит о восстановлении данных. SQL Server поддерживает частичные резервные копии, т. е. отдельно создаются резервные копии файловых групп только для чтения и файловых групп для чтения и записи. Затем можно использовать эти резервные копии для выполнения поэтапного восстановления, которое позволит привести базу данных обратно в оперативный режим.

Грамотное использование файловых групп повышает производительность работы базы данных. При создании объекта можно указать файловую группу для его размещения. Если файловая группа не указана, то будет использоваться файловая группа по умолчанию. Любую файловую группу

(кроме файловых групп только для чтения) можно назначить файловой группой по умолчанию. Создавая объекты, сильно конкурирующие за дисковое пространство, можно разнести их по разным файловым группам. Таким образом, снижение конкуренции будет приводить к повышению производительности.

Если файловая группа содержит несколько файлов, SQL Server записывает данные во все файлы одновременно с помощью так называемой «стратегии пропорционального заполнения». Файлы, которые имеют одинаковый размер, будут иметь одинаковое количество записанных в них данных, обеспечивая их заполнение с постоянной скоростью. Файлы, которые имеют различные размеры, будут иметь различное количество данных. Это позволяет использовать файловую группу для реализации простой формы чередования. Можно создать файловую группу, содержащую два или более файлов, каждый из которых находится на отдельном диске. Когда SQL Server пишет в файловую группу, он может использовать отдельный канал ввода / вывода для записи на каждый диск одновременно (параллельно), что приводит к сокращению времени записи.

В первую очередь следует использовать файловые группы для улучшения управляемости, используя саму конфигурацию устройства хранения для повышения производительности операций ввода/вывода. Если существуют проблемы (или недоступно аппаратное обеспечение RAID), использование файловых групп для размещения файлов данных на физических дисках может стать эффективной альтернативой.

Создание файловых групп

Можно создать дополнительные файловые группы и назначить им файлы при создании базы данных или добавить новые файловые группы и файлы к существующей базе данных позднее.

В следующем примере создается база данных Sales.

Первичная файловая группа содержит один файл sales.mdf, две пользовательских файловых группы данных содержат по два файла каждая: файловая группа *Transactions* файлы sales_tran1.ndf и sales_tran2.ndf, а файловая группа *Archive* – файлы sales_archive1.ndf и sales_archive2.ndf.

Журнал транзакций содержит один файл, но он не принадлежит ни к одной из файловых групп!

```
CREATE DATABASE Sales
ON  PRIMARY
     (NAME = 'Sales', FILENAME = 'D:\Data\sales.mdf',
     SIZE = 5MB, FILEGROWTH = 1MB),
FILEGROUP Transactions
     (NAME = 'SalesTrans1',
          FILENAME = 'M:\Data\sales_tran1.ndf',
          SIZE = 50MB, FILEGROWTH = 10MB),
     (NAME = 'SatesTrans2',
```

```
          FILENAME = 'N:\Data\sales_tran2.ndf',
          SIZE = 50MB, FILEGROWTH = 10MB),
FILEGROUP Archive
     (NAME = 'HistoricData1',
          FILENAME = 'O:\Data\sales_archive1.ndf',
          SIZE = 200MB, FILEGROWTH = 10MB),
     (NAME = 'HistoricData2',
          FILENAME = 'P:\Data\sales_archive2.ndf',
          SIZE = 200MB, FILEGROWTH = 10MB)
LOG ON
     (NAME = 'Sales_log',
          FILENAME = 'L:\Logs\sales.ldf',
          SIZE = 10MB , FILEGROWTH = 1MB);
```

Следующий код сделает файловую группу *Transactions* файловой группой по умолчанию, а файловую группу *Archive* файловой группой только для чтения:

```
ALTER DATABASE Sales
     MODIFY FILEGROUP Transactions DEFAULT;
ALTER DATABASE Sales
     MODIFY FILEGROUP Archive READONLY;
```

Для добавления файловых групп в существующей базе данных, можно использовать инструкцию ALTER DATABASE.

2.3 Перемещение файлов и баз данных

Может потребоваться переместить файлы базы данных или даже целые базы данных. Рассмотрим, как можно это сделать, а также как можно скопировать базу данных целиком.

Перемещение файлов и баз данных

Можно перемещать файлы базы данных в другое место, используя оператор ALTER DATABASE или с помощью SSMS.

Перед тем, как перемещать файлы базы данных, нужно перевести базу данных в автономный режим.

При перемещении файлов базы данных нужно использовать логическое имя файла, которое задается файлу при создании базы данных. Узнать логическое имя файла можно с помощью системного представления sys.database_files.

В следующем примере показано, как перевести БД AdventureWorks в автономный режим, переместить файл данных и вернуть БД в режим online:

```
ALTER DATABASE AdventureWorks SET OFFLINE;
               // Move the files on the file system
ALTER DATABASE AdventureWorks
     MODIFY FILE (NAME = AWDataFile,
     FILENAME = 'C:\AWDataFile.mdf');
ALTER DATABASE AdventureWorks SET ONLINE;
```

Отсоединение (Detaching) и присоединение (Attaching) баз данных

Для подключения к конкретному экземпляру SQL Server или отключения базы данных можно использовать функциональность Detaching и Attaching, которую предоставляет SQL Server. Это широко используемый метод для перемещения базы данных из одного экземпляра в другой.

Отсоединить базу данных от экземпляра SQL Server можно с помощью среды SSMS или системной хранимой процедурой sp_detach_db.

Отсоединение базы данных не удаляет данные из ее файлов. Удаляются записи метаданных об этой базе данных из системных баз данных на экземпляре SQL Server. Затем отсоединенная база данных больше не отображается в списке баз данных в среде SSMS или в результатах системного представления sys.databases. После отсоединения базы данных ее файлы можно перемещать или копировать, а затем присоединить к другому экземпляру SQL Server.

Не все базы данных могут быть отсоединены. Базы данных, которые настроены для репликации, зеркальные или в подозрительном состоянии не могут быть отсоединены.

Нельзя отсоединить базу данных, если есть открытые соединения. Только после закрытия всех открытых соединений SQL Server позволит выполнить операцию Detach. Использование среды SSMS во время этой операции предлагает возможность принудительного отключения (закрытия) всех соединений.

Присоединение баз данных можно выполнить в среде SSMS или с помощью оператора `CREATE DATABASE … FOR ATTACH`.

Можно использовать системные хранимые процедуры sp_attach_db и sp_attach_single_file_db. Их устаревший синтаксис заменен на параметр FOR ATTACH инструкции CREATE DATABASE. Также обратите внимание, что нет эквивалентной замены для процедуры sp_detach_db.

Общей проблемой повторного подключения базы данных является то, что пользователи базы данных могут лишиться прав доступа.

---

3 РЕЗЕРВНОЕ КОПИРОВАНИЕ: ПЛАНИРОВАНИЕ И РЕАЛИЗАЦИЯ

Одним из наиболее важных аспектов роли DBA является обеспечение регулярного копирования данных, чтобы в случае сбоя можно было восстановить БД. Несмотря на то, что компьютерная индустрия уже на протяжении десятилетий знает о необходимости надежных стратегий резервного копирования, трагические истории потери данных по-прежнему остаются обычным явлением.

3.1 Модели восстановления SQL сервера

SQL сервер поддерживает три типа моделей восстановления базы данных. Все модели сохраняют данные в случае аварии, но существуют важные различия, которые необходимо знать при выборе модели восстановления для базы данных [4]. Описание моделей восстановления приведено в Таблице 5. У каждой базы данных может быть своя модель восстановления. Выбор модели осуществляется в окне Свойства БД на странице Параметры.

Таблица 5. Описание моделей восстановления в SQL Server

Модель восстановления	**Описание**
Простая (Simple)	Предназначена для восстановления до точки последней архивации. Стратегия для данной модели должна включать полные и разностные операции резервного копирования. Включив простую модель восстановления, нельзя выполнять резервное копирование журналов транзакций. Автоматически в контрольной точке происходит усечение журнала (очистка всех неактивных записей транзакций), чтобы минимизировать пространство. Эта модель идеально подходит для большинства системных БД.
Полная (Full)	Предназначена для восстановления БД до точки сбоя или на определенный момент времени. При использовании данной модели протоколируются все операции, включая массовые операции и массовую загрузку данных. Стратегия должна включать следующие архивы: полные, разностные и архивы журнала транзакций (или только полные архивы и архивы журнала транзакций), т.е. требуется резервное копирование журнала. Исключает потерю данных из-за поврежденного или отсутствующего файла данных.
Неполное протоколирование (Bulk Logged)	Эта модель сокращает пространство, занимаемое журналом транзакций, сохраняя бо**л**ьшую часть функциональности полной модели восстановления. Выполняется минимальное протоколирование массовых операций и массовых загрузок без контроля отдельных операций, что может повысить производительность операций массового копирования. Если сбой произойдет прежде, чем будет выполнена полная или разностная архивация, массовые операций и массовые загрузки придется повторить вручную. Стратегия резервного копирования для данной модели должна включать те же архивы, что и для полной модели.

Контрольные точки (Checkpoint)

SQL Server Database Engine выполняет изменения страниц базы данных в памяти (в буферном кэше) и не записывает эти страницы на диск сразу же после каждого изменения. Вместо этого Database Engine периодически создает контрольную точку на каждой базе данных. Именно в контрольной точке он записывает из памяти на диск текущие страницы, измененные в памяти (известные как измененные незафиксированные страницы), а также записывает сведения в журнал транзакций.

Компонент Database Engine поддерживает четыре типа контрольных точек: автоматические, косвенные, ручные и внутренние.

- **Автоматические** контрольные точки (Automatic checkpoints). Создаются с помощью системной хранимой процедуры sp_configure, которой передается параметр конфигурации сервера recovery interval:

  ```
  EXEC sp_configure "recovery interval«,»seconds" **.
  ```

  В соответствии с верхним пределом времени, указанном в этом параметре, контрольные точки автоматически выдаются в фоновом режиме. Автоматические контрольные точки выполняются до их завершения. Автоматические контрольные точки регулируются в зависимости от числа необработанных записей и от того, обнаруживает ли компонент Database Engine увеличение задержки записи более чем на 20 миллисекунд.
- **Косвенные** контрольные точки (Indirect checkpoints). Создаются с помощью инструкции ALTER DATABASE:

  ```
  ALTER DATABASE … SET TARGET_RECOVERY_TIME
         =target_recovery_time { SECONDS | MINUTES }.
  ```

  Выдаются в фоновом режиме для обеспечения соответствия пользовательскому целевому времени восстановления для конкретной базы данных. Целевое время восстановления по умолчанию равно 0, что приводит к использованию эвристики автоматических контрольных точек в базе данных. Если была использована инструкция ALTER DATABASE для установки TARGET_RECOVERY_TIME в значение > 0, то используется это значение, а не интервал восстановления, указанный для экземпляра сервера.
- **Ручные** контрольные точки (Manual checkpoints). Выдаются при выполнении команды Transact-SQL CHECKPOINT:

  ```
  CHECKPOINT [ checkpoint_duration ]
  ```

  Ручная контрольная точка срабатывает в текущей базе данных для конкретного соединения. По умолчанию ручная контрольная точка выполняется до ее завершения. Регулирование работает так же, как и для автоматической контрольной точки. При необходимости параметр checkpoint_duration указывает требуемое время в секундах для завершения контрольной точки.

- **Внутренние** контрольные точки (Internal checkpoints). Выдаются различными операциями сервера, такими как резервное копирование и создание моментального снимка базы данных, для обеспечения соответствия образа диска текущему состоянию журнала.

3.2 Резервное копирование Баз Данных и Журналов

Типы резервных копий

SQL Server поддерживает несколько типов резервных копий, которые можно комбинировать для реализации правильной стратегии резервного копирования и восстановления для конкретной базы данных на основе бизнес-требований и целей восстановления. Описание типов резервных копий приведено в Таблице 6.

Таблица 6. Описание типов резервных копий в SQL Server

Тип резервной копии	**Описание**
Полная (**Full**)	В полную резервную копию БД включаются все объекты, системные таблицы и данные, а также фрагменты журналов транзакций, соответствующие времени архивации. Полная резервная копия позволяет полностью восстановить БД на момент завершения резервного копирования.
Разностная (**Differential**)	Разностные резервные копии предназначены для архивации данных, которые изменились со времени последнего резервного копирования. При резервном копировании сохраняются только изменения, поэтому архивация занимает меньше времени, и ее можно выполнять чаще. В разностные копии также включаются фрагменты журналов транзакций, необходимые для восстановления БД после завершения резервного копирования
Копия Журнала Транзакций (**Transaction Log**)	При резервном копировании журнала транзакций в архиве сохраняются изменения, произошедшие со времени последнего резервного копирования журнала транзакций, а затем выполняется его усечение, в ходе которого стираются выполненные или отмененные транзакции. В отличие от полных или разностных архивов, в архиве журнала транзакций записано состояние журнала на момент начала операции резервного копирования (а не на момент его завершения).
Копия файлов и файловых групп (**File/File Group**)	Эти резервные копии позволяют архивировать не всю БД, а только указанные файлы и файловые группы. Резервные копии этого типа используют при работе с большими БД, чтобы сэкономить время. Архивация файлов и файловых групп имеет ряд особенностей. Одновременно с ней надо архивировать и журнал транзакций. Поэтому данным способом НЕ выполняют архивацию, если включен параметр Усечение журнала на контрольной точке (Truncate Log On Checkpoint). Если объекты БД охватывают несколько файлов или файловых групп, надо выполнить архивацию ВСЕХ этих файлов или файловых групп.

Тип резервной копии	Описание
Частичная (**Partial**)	Содержит данные только из некоторых файловых групп БД, включая данные первичной файловой группы, все файловые группы, доступные для чтения/записи, а также любые дополнительно указанные файлы, доступные только для чтения. Частичные резервные копии могут оказаться полезны в тех случаях, когда необходимо исключить файловые группы только для чтения. Частичная резервная копия базы данных, доступной только для чтения, содержит только первичную файловую группу.
Копия заключительного фрагмента журнала (Tail-log Backup)	В резервную копию заключительного фрагмента журнала попадают все записи, резервная копия которых еще не была создана (заключительный фрагмент журнала, «хвост»), что позволяет предотвратить потерю работы и сохранить неповрежденную цепочку журналов. Для восстановления базы данных SQL Server на последний момент времени необходимо предварительно выполнить резервное копирование заключительного фрагмента журнала ее транзакций. Заключительный фрагмент журнала становится последней рассматриваемой частью резервной копии в плане восстановления базы данных.
Копия только для копирования (Copy Only)	Специальная резервная копия, независимая от обычной последовательности резервных копий SQL Server. Обычно создание резервного копирования приводит к изменению БД и влияет на то, как будут восстанавливаться последующие резервные копии. Однако иногда приходится выполнять резервное копирование БД для особых нужд, когда это не сказывается на общем процессе резервного копирования и восстановления. Этой цели и служат резервные копии только для копирования (БД или журналов транзакций).

Выполнение резервного копирования подробно описано в электронной документации по SQL Server 2014 на сайтах Microsoft [1-3]. Там же можно найти примеры реализации различных стратегий резервного копирования.

3.3 Использование параметров резервного копирования

Наличие большого набора типов резервных копий обеспечивают широкий выбор вариантов, которые могут помочь оптимизировать стратегию резервного копирования, включая возможности сжатия и шифрования.

Сжатие резервных копий

Резервные файлы могут быстро стать очень большими, поэтому SQL Server позволяет сжимать их. Можно задать поведение сжатия резервных копий по умолчанию, а также переопределить этот параметр для отдельных резервных копий. Для сжатых резервных копий применяются следующие ограничения:

- В наборе носителей не допускается одновременное существование сжатых и несжатых резервных копий.
- Резервные копии на базе Windows не могут совместно использовать набор носителей со сжатыми резервными копиями SQL Server.

- Можно создавать сжатые резервные копии только в выпусках Enterprise, Business Intelligence и Standard Editions SQL Server 2014. А восстановить сжатые резервные копии можно в любом выпуске SQL Server 2008, 2008 R2, 20012 или 2014.

Страницы свойств сервера можно использовать для просмотра и настройки сжатия резервных копий по умолчанию.

Для сжатия резервной копии, можно использовать параметр WITH COMPRESSION в инструкции BACKUP, как в приведенном ниже примере сжатия резервной копии БД AdventureWorks:

```
BACKUP DATABASE AdventureWorks
     TO DISK = 'R:\Backups\AW_Comp.bak'
     WITH COMPRESSION;
```

Если настройка резервных копий по умолчанию со сжатием, а вы хотите это отменить, используйте опцию `NO_COMPRESSION`.

Уровень сжатия, который может быть достигнут, полностью зависит от того, какие данные хранятся в базе данных. Некоторые данные сжимаются хорошо, другие нет.

Можно вычислить коэффициент сжатия резервной копии, используя значения столбцов backup_*size* и *compressed_backup_size* из таблицы журнала *backupset* для резервной копии следующим образом.

```
SELECT backup_size/compressed_backup_size
FROM msdb..backupset;
```

Коэффициент сжатия 3:1 означает, что экономится около 66% места на диске.

Сжатие резервных копий можно использовать в БД, которая шифруется с помощью прозрачного шифрования (TDE), хотя степень сжатия будет минимальной.

По умолчанию сжатие существенно повышает загрузку на центральный процессор (ЦП), что может помешать выполнению других операций. Поэтому может потребоваться создать сжатые резервные копии с низким приоритетом в сеансе, для которого использование ЦП ограничивается регулятором ресурсов.

Шифрование резервных копий

Для резервных копий основополагающим требованием является обеспечение защиты данных от аппаратных сбоев и пр. Однако данные в резервной копии могут быть конфиденциальными, поэтому необходимо убедиться, что носитель резервных копий защищен от несанкционированного доступа. В большинстве организаций это может быть выполнено путем хранения носителей резервных копий в местах с защищенной файловой системой.

В ситуации, когда данные в резервной копии требуют дополнительной безопасности, можно шифровать резервные копии таким образом, чтобы они могли быть восстановлены только на экземпляре SQL Server, содержащем ключ шифрования. Резервное копирование с шифрованием в SQL Server основывается на алгоритмах стандарта шифрования, включая AES 128, AES 192, AES 256 и Triple DES. Для шифрования резервной копии необходимо указать алгоритм, который вы хотите использовать, и сертификат (или асимметричный ключ), который может использоваться для шифрования данных.

Чтобы использовать шифрование резервных копий:

1. Создайте главный ключ базы данных в базе данных master. Это симметричный ключ, который используется для защиты всех других ключей шифрования и сертификатов в базе данных.

2. Создайте сертификат или асимметричный ключ для шифрования резервной копии. Можно создать сертификат или асимметричный ключ в экземпляре ядра базы данных SQL Server с помощью оператора CREATE CERTIFICATE или CREATE ASYMMETRIC KEY. Обратите внимание, что асимметричные ключи должны находиться в поставщике расширенного управления ключами (EKM).

3. Выполняйте резервное копирование с помощью параметра ENCRYPTION (или выбор шифрования в диалоговом окне Резервное копирование базы данных), а также с указанием алгоритма и сертификата (или асимметричного ключа), который будет использоваться. При использовании диалогового окна резервного копирования, необходимо выбрать опцию для резервного копирования на новый набор носителей.

Необходимо создать резервную копию мастер-ключа и ключей шифрования в безопасном месте (отдельно от расположения носителя резервного копирования), чтобы вы могли восстановить базу данных на другой экземпляр SQL Server в случае полного отказа сервера.

Следующий пример кода создает резервную копию базы данных AdventureWorks с использованием алгоритма шифрования AES 128 и сертификата BackupCert:

```
BACKUP DATABASE AdventureWorks
     TO DISK = 'R:\Backups\AW_Encrypt,bak'
     WITH FORMAT, INIT,
     ENCRYPTION(   ALGORITHM=AES_128,
                   SERVER CERTIFICATE = [BackupCert])
```

Можно создавать зашифрованные резервные копии только в выпусках Enterprise, Business Intelligence и Standard Editions SQL Server 2014, а восстанавливать в любом выпуске SQL Server 2014.

Хранение и политика тестирования резервных копий

Независимо от того, сколько создается резервных копий, важно убедиться, что они читаемы и восстанавливаемые, в противном случае вся система резервного копирования является ущербной. Важно также иметь возможность запрашивать информацию о резервных копиях. Поэтому надо определиться с политиками хранения и тестирования резервных копий.

Планирование хранения резервной копии должно быть частью общей стратегии, которое требует тщательности, аккуратности.

Надо предусмотреть:

- Наличие комбинации резервных копий, необходимых для восстановления базы данных.
- Выполнение требований архивации.
- Синхронизацию с проверками базы данных.
- Наличие доступного и безопасного места для хранения резервных копий.
- Аппаратные средства, необходимые для восстановления резервных копий
- Полноту резервных копий
- Архивные требования

Параметры для обеспечения целостности резервной копии

Резервное копирование SQL Server включает параметры для обеспечения целостности резервных копий, снижая риск того, что резервные копии являются непригодными для использования и для восстановления.

Параметр MIRROR TO. Зеркальные наборы носителей являются копией резервного набора носителей, которые можно создать параллельно с основной операцией резервного копирования. Он состоит из двух-четырех зеркал, каждое из которых содержит весь набор носителей. Требуется настроить каждое зеркало.

Наличие не одной, а нескольких резервных копий может увеличить доступность данных. Однако важно понимать, что зеркалирование набора носителей подвергает вашу систему более высокому уровню риска сбоя оборудования, так как неисправности любого из устройств резервного копирования приводит к сбою всей операции резервного копирования.

Можно создать зеркальный резервный набор с помощью параметра MIRROR TO инструкции BACKUP:

```
BACKUP DATABASE AdventureWorks
     TO DISK = 'R:\Backups\AW.bak'
     MIRROR TO DISK = 'Q:\ Backups\AW_M.bak'
     WITH FORMAT, INIT;
```

Зеркальная функциональность набора носителей доступна только в SQL Server Enterprise Edition

Параметр WITH CHECKSUM. Позволяет выполнять операцию проверки контрольной суммы в течение всего потока резервного копирования и записи значения до конца резервной копии.

Параметр WITH CHECKSUM проверяет информацию на уровне страницы. Это гарантирует, что резервная копия находится в хорошем состоянии, а во время операции восстановления это гарантирует, что носитель резервной копии сам не был поврежден во время копирования файлов или передачи.

Однако это потребляет чуть больше ресурсов ЦП во время процесса резервного копирования, чем резервное копирование без расчета контрольной суммы.

Можно настроить SQL Server для оценки значения контрольной суммы во время операций восстановления или во время операций резервного копирования проверки с помощью команды RESTORE VERIFYONLY.

Пример использования контрольной суммы:

```
BACKUP DATABASE AdventureWorks
     TO DISK = 'R:\Backups\AW.bak'
     WITH CHECKSUM;
```

Проверка резервных копий. Для проверки резервной копии, можно использовать инструкцию RESTORE VERIFYONLY, которая проверяет резервную копию на допустимость, но не восстанавливает ее. Оператор выполняет следующие проверки:

- Резервный набор данных полный.
- Все тома читаются.
- Идентификаторы страницы являются правильными.
- Контрольная сумма действительна (если она присутствует на носителе).
- Имеется достаточное пространство на целевых устройствах.

Значение контрольной суммы может быть проверено только в том случае, если резервная копия была выполнена с параметром WITH CHECKSUM.

Инструкция RESTORE VERIFYONLY аналогична инструкции RESTORE и поддерживает подмножество аргументов.

Проверка резервной копии может быть выполнена следующим оператором:

```
RESTORE VERIFYONLY
     FROM DISK = 'R:\Backups\AW.bak'
```

Можно также выполнить шаги проверки с помощью задачи резервного копирования базы данных в среде SSMS.

Рекомендуется: выполнить, проверку резервных копий на другой системе, а не там, где была сделана резервная копия. Это позволит избежать ситуации, когда резервная копия доступна только для чтения на исходном оборудовании.

---

4 ВОССТАНОВЛЕНИЕ БАЗ ДАННЫХ

SQL Server предоставляет множество возможностей для восстановления баз данных. Во-первых, это восстановление базы данных целиком: оно может занимать довольно много времени (зависит от размера БД и скорости жестких дисков). Во-вторых, восстановление отдельных файловых групп, либо файлов, если ваша БД состоит из нескольких файлов и файловых групп. В этом случае, есть возможность восстановления только поврежденных частей БД. Эти два вида восстановления БД используются довольно часто.

В-третьих, в SQL Server 2005 появилась возможность восстановления отдельных страниц БД: в этом случае из резервной копии будут восстановлены только указанные страницы. Такое восстановление будет особенно актуально, если DBCC CHECKDB найдет несколько поврежденных страниц в какой-нибудь огромной таблице, хранящейся в файле очень большого размера.

4.1 Обзор процесса восстановления

Если необходимо восстановить базу данных, важно иметь хороший план, чтобы избежать дальнейших потерь. После того, как вы завершили предварительный шаг в попытке создать резервную копию заключительного фрагмента журнала транзакций, наиболее важно определить, какие имеются резервные копии базы данных для восстановления и в каком порядке.

Восстановление базы данных SQL Server состоит из трех фаз: копирование данных, стадия повтора и стадия отмены. Сочетание фазы повтора и фазы отмены обычно называют восстановление базы данных.

Копирование данных

Стадия копирования данных является, как правило, самой длинной в восстановлении базы данных. Во-первых, файлы данных из базы данных должны быть восстановлены из резервных копий. До того, как будут восстановлены все страницы данных, процесс восстановления считывает заголовок резервной копии, и SQL Server восстанавливает требуемые файлы данных и журналов. Если инициализация экземпляра файла (instant file initialization, IFI) не была включена путем предоставления прав учетной записи службы SQL Server, перезапись файлов данных может занять значительное время.

После восстановления данных и журналов, файлы данных восстанавливаются из полной резервной копии. Страницы данных извлекаются по порядку из резервной копии и записываются в файлы данных. Файлы журнала должны быть обнулены, прежде чем они могут быть использованы. Этот процесс также может занять значительное время, если файлы журнала большие.

Если разностная резервная копия также используется, SQL Server перезаписывает экстенты в файлах данных, используя те экстенты, которые содержатся в разностной резервной копии.

Стадия повтора (Redo Phase)

В начале фазы повтора SQL Server извлекает данные из журнала транзакций. В простой модели восстановления эти данные извлекаются либо из полной резервной копии, или из разностной резервной копии. В модели восстановления с неполным или полным протоколированием эти сведения дополняются содержимым резервных копий журнала транзакций, которые были получены после полного и разностного резервного копирования БД.

В фазе повтора, SQL Server «накатывает» все изменения, которые содержатся в деталях журнала транзакций вплоть до точки восстановления. Как правило, точка восстановления – это самое позднее время, для которого существуют транзакции в журнале.

Стадия отмены (Undo Phase)

Журнал транзакций будет, вероятно, включать информацию об операциях, которые не были завершены в точке восстановления (на момент сбоя). В стадии отмены SQL Server выполняет откат всех этих незавершенных транзакций.

Поскольку эта стадия включает откат незафиксированных транзакций и перевод базы данных в онлайн, больше никакие резервные копии не могут быть восстановлены.

На этапе отката Enterprise edition переводит базу данных в оперативный режим для доступа пользователям. Эта возможность называется функцией быстрого восстановления. Запросы, которые пытаются получить доступ к данным, по-прежнему блокируются до завершения стадии отката. Это может вызвать время ожидания выполнения транзакции, но означает, что пользователи уже могут получить доступ к базе данных.

В общем нельзя перевести базу данных в оперативный режим, пока она не будет восстановлена. Единственным исключением является вариант быстрого восстановления, который позволяет пользователям получить доступ к базе данных в то время, как стадия отката продолжается.

Восстановление не только происходит во время выполнения команды RESTORE. Восстановление базы данных также будет происходить, если база данных переводится в автономный режим, а затем возвращается обратно в состояние online. Тот же самый процесс восстановления происходит, когда перезагружается SQL Server.

4.2 Восстановление Баз данных

Большинство операций восстановления включает в себя восстановление полной резервной копии, затем следуют разностная резервная копия и последовательность резервных копий журнала транзакций.

Восстановление полной резервной копии

Восстановить базу данных можно либо с помощью среды SSMS, либо инструкцией RESTORE DATABASE.

1. Простейший сценарий восстановления

Простейший сценарий восстановления – восстановить базу данных из одной полной резервной копии. Если никакие последующие разностные или резервные копии журналов транзакций не должны быть применены, можно использовать параметр RECOVERY, чтобы указать, что SQL Server должен завершить процесс восстановления БД и перевести ее в режим online.

Если необходимо восстановить дополнительные резервные копии, можно предотвратить завершение восстановления, указав параметр WITH NORECOVERY.

Если не указать один из этих параметров, SQL Server использует параметры по умолчанию.

В следующем примере база данных AdventureWorks восстанавливается из полной резервной копии AW.bak:

```
RESTORE DATABASE AdventureWorks
     FROM DISK = 'R:\Backups\AW.bak';
```

2. Замена существующей базы данных.

SQL Server не позволяет восстановить резервную копию базы данных поверх существующей базы данных, если вы не выполнили резервную копию заключительного фрагмента журнала базы данных.

Если вы все-таки попытаетесь это сделать с помощью среды SSMS, SQL Server будет выдавать предупреждение и сам попробует создать резервную копию заключительного фрагмента журнала. Если необходимо выполнить операцию восстановления, а резервной копии заключительного фрагмента журнала нет, необходимо указать параметр WITH REPLACE.

В следующем примере кода, существующая база данных AdventureWorks заменяется:

```
RESTORE DATABASE AdventureWorks
     FROM DISK = 'R:\Backups\AW.bak
     WITH REPLACE';
```

Параметр WITH REPLACE должен использоваться с осторожностью, поскольку это может привести к потере данных.

---

3. Восстановление файлов базы данных в другое место.

При восстановлении базы данных с другого сервера, возможно, потребуется разместить файлы базы данных в разных местах. Также может потребоваться сделать это при копировании базы данных с помощью резервного копирования и восстановления. Параметр WITH MOVE позволяет указать новые местоположения файлов.

В этом примере база данных AdventureWorks восстанавливается с другого сервера. В инструкции RESTORE указано местоположение источника для набора носителей, а также новые места для каждого файла базы данных. Обратите внимание, что опция MOVE требует логическое имя файла, а не исходный физический путь к нему.

```
RESTORE DATABASE AdventureWorks
FROM DISK = 'R:\Backups\AW.bak'
WITH MOVE 'AdventureWorks_Data'
          TO 'Q:\Data\AdventureWorks.mdf',
MOVE 'AdventureWorks_Log'
          TO 'U:\Logs\ AdventureWorks.ldf';
```

4. Восстановление базы данных в режиме ожидания.

SQL Server предоставляет возможность просматривать содержимое базы данных, которая не была восстановлена, с помощью параметра WITH STANDBY, но не с параметром WITH NORECOVERY. После восстановления базы данных с помощью параметра WITH STANDBY, можно еще применить дополнительные резервные копии журнала транзакций к базе данных. Опция STANDBY обычно используется для поддержки сценариев доставки журналов, в которых дополнительная копия базы данных синхронизируется путем повторного применения транзакций в журнале транзакций базы данных источника. Вы также можете использовать параметр STANDBY для включения проверки данных в БД, если не хотите переводить базу в оперативный режим.

Восстановление разностной резервной копии

В стратегии дифференциальной резервной копии необходимо восстановить первоначальную полную резервную копию базы данных, а затем восстановить последнюю разностную резервную копию.

1. Восстановление БД при восстановлении нескольких резервных копий.

Как уже говорилось ранее, процесс восстановления в SQL Server имеет решающее значение для поддержания целостности транзакций. Это требует, чтобы все операции, которые были завершены до сбоя, фиксировались в базе данных, а все операции, которые не завершились, откатились.

Команда RESTORE включает параметр, указывающий WITH RECOVERY или WITH NORECOVERY. Опция WITH RECOVERY является параметром по умолчанию и может не указываться. Это гарантирует, что база данных переводится в оперативный режим сразу после восстановления из полной

резервной копии. Однако, если ваша стратегия резервного копирования требует, чтобы после полной резервной копии базы данных были восстановлены дополнительные резервные копии, важно выбрать правильный вариант для каждой команды RESTORE. Процесс в большинстве случаев прост. Все операции восстановления должны быть с опцией NORECOVERY за исключением последней, которая должна выполняться с параметром WITH RECOVERY.

Нет никакого способа восстановить дополнительные резервные копии после восстановления с параметром WITH RECOVERY. Если вы случайно выполнить резервное копирование с помощью параметра WITH RECOVERY, необходимо перезапустить всю последовательность восстановления.

2. Восстановление базы данных из разностной резервной копии.

Команда для восстановления дифференциальной резервной копии идентична восстановлению полной резервной копии базы данных. Разностные резервные копии часто добавляются в тот же файл в качестве полной резервной копии (в этом случае вам нужно указать конкретный файл из набора носителей, который требуется восстановить).

В этом примере база данных AdventureWorks восстанавливается из первого файла в наборе, содержащем полную резервную копию. Этот набор носителей хранится в файле R:\Backups\AW.bak. Второй файл в наборе носителей является первой разностной резервной копией, но во второй разностной резервной копии (в третьем файле) эти изменения также содержатся. Таким образом, необходимо только восстановить содержимое третьего файла (вторая инструкция RESTORE):

```
RESTORE DATABASE AdventureWorks
     FROM DISK = 'R:\Backups\AW.bak'
     WITH FILE = 1, NORECOVERY;
RESTORE DATABASE AdventureWorks
     FROM DISK = 'R:\Backups\AW.bak'
     WITH FILE = 3,  RECOVERY;
```

Если оба – полные и дифференциальные – резервные наборы находятся на одном и на том же носителе, SSMS автоматически выбирает необходимые резервные копии в окне диалога восстановления базы данных и гарантирует, что применяются правильные параметры восстановления.

Восстановление резервной копии журнала транзакций

Журналы транзакций для базы данных можно восстановить в среде SSMS или с помощью инструкции RESTORE LOG. Необходимо восстановить все файлы журналов (за исключением последнего) с помощью параметра WITH NORECOVERY, а затем восстановить последний файл журнала (который часто является резервной копией заключительного фрагмента журнала) с помощью параметра WITH RECOVERY.

При использовании стратегии резервного копирования, которая включает в себя резервное копирование журнала транзакций, вы должны начать

восстановление последней полной резервной копии базы данных, а затем последней разностной резервной копии, если таковая существует. А затем необходимо восстановить все журналы транзакций, созданные после последней полной или разностной резервной копии в хронологическом порядке. Любой перерыв в цепочке журналов транзакций вызовет ошибку и потребует перезапуска восстановления с самого начала.
В следующем примере база данных AdventureWorks была испорчена, но файл журнала доступен, так что резервная копия заключительного фрагмента журнала хранится в AW-TailLog.bak:

```
     -- Restore last full database backup
RESTORE DATABASE AdventureWorks
     FROM DISK = 'R:\Backups\AW.bak'
     WITH FILE = 1, NORECOVERY;
     -- Restore last differential backup
RESTORE DATABASE AdventureWorks
     FROM DISK = 'R:\Backups\AW.bak'
     WITH FILE = 3,  NORECOVERY;
     -- Restore planned log backups
RESTORE LOG AdventureWorks
     FROM DISK = 'R:\Backups\AW.bak'
     WITH FILE = 4,  NORECOVERY;
RESTORE LOG AdventureWorks
     FROM DISK = 'R:\Backups\AW.bak'
     WITH FILE = 5,  NORECOVERY;
     -- Restore tail-log backup
RESTORE LOG AdventureWorks
     FROM DISK = 'R:\Backups\AW-TailLog.bak'
     WITH RECOVERY;
```

В предыдущем примере, файл журнала был доступен после сбоя базы данных, так что резервное копирование хвоста журнал было возможно. Это позволяет восстановить БД до точки отказа. Если бы файл журнала не был доступен, последнее плановое резервное копирование журнала транзакций (резервный набор 5 в AW.bak) был бы восстановлен с помощью опции RECOVERY, а все транзакции после резервной копии были бы потеряны.

4.3 Расширенные сценарии восстановления

В разделе 4.2 были описаны широко распространенные сценарии восстановления. Но существуют более сложные сценарии восстановления, к использованию которых в своей работе DBA должен быть подготовлен:

- восстановление резервной копии файла или файловой группы (в том числе поэтапное восстановление);
- восстановление зашифрованной резервной копии;
- восстановление отдельных страниц данных;
- восстановление системных баз данных.

Восстановление резервной копии файла или файловой группы

Бывает гораздо быстрее восстановить один файл или файловую группу, чем всю базу данных. Не нужно создавать резервные копии определенных файлов или файловых групп для того, чтобы восстановить их, поскольку SQL Server может извлекать файлы конкретной базы данных из полной или разностной резервной копии.

Для восстановления файла или файловой группы надо выполнить следующие действия:

1. Создайте резервную копию заключительного фрагмента активного журнала транзакций. (Если вы не можете сделать это, так как журнал был поврежден, необходимо восстановить всю базу данных или восстановить ее на предшествующий момент времени.)
2. Восстановите каждый поврежденный файл из последней резервной копии этого файла.
3. Восстановите последнюю разностную резервную копию файла, если таковые имеются, для каждого восстановленного файла.
4. Восстановите резервные копии журналов транзакций в последовательности, начиная с резервной копии, которая охватывает самый старый из восстановленных файлов и заканчивая резервной копией хвоста журнала, созданной на шаге 1.
5. Восстановите базу данных.
   Необходимо восстановить резервные копии журнала транзакций, созданные после резервных копий файлов, чтобы перевести базу данных обратно в согласованное состояние. Резервные копии журналов транзакций можно быстро накатить, поскольку применяются только изменения, которые касаются восстановленных файлов или файловых групп. Неповрежденные файлы не копируются. Однако вам все еще нужно обработать всю цепочку резервных копий журналов.

Поэтапное восстановление

Как уже говорилось, если база данных чрезвычайно велика, то для хранения неактивных данных можно использовать файловые группы только для чтения и использовать частичную стратегию резервного копирования. Т.е. резервное копирование файловой группы только для чтения выполняется один раз, и только файловые группы для чтения/записи включаются в последующие резервные копии, что значительно сокращает время, необходимое для создания полной или разностной резервной копии.

Одним из преимуществ частичной стратегии резервного копирования файловых групп является то, что она позволяет выполнить поэтапное восстановление. В поэтапном восстановлении сначала можно восстановить файловые группы для чтения/записи и сделать их доступными пользователям для выполнения запросов (до завершения восстановления файловых групп только для чтения).

Для выполнения поэтапного восстановления надо:

1. Восстановить последнюю частичную полную резервную копию базы данных, с указанием, что файловые группы для чтения/записи должны быть восстановлены, используя параметр PARTIAL, чтобы указать: файловые группы только для чтения будут восстановлены отдельно.

2. Восстановить последнюю частичную разностную резервную копию и резервные копии файлов журнала, если они существуют. Используйте параметр RECOVERY в последней операции RESTORE для восстановления базы данных. Данные в файловых группах для чтения/записи теперь доступны.

3. Восстановить резервные копии каждой файловой группы только для чтения с помощью параметра RECOVERY, чтобы перевести их в режим online.

Восстановление зашифрованной резервной копии

Вы можете восстановить зашифрованную резервную копию для любого экземпляра SQL Server, на котором размещен сертификат или ключ, который использовали для шифрования резервной копии.

Это означает, что если нужно восстановить зашифрованную резервную копию на том же сервере, на котором она создавалась, можно ее восстановить, используя ту же процедуру, что и для обычной (незашифрованной) резервной копии, так как сертификат или ключ есть в экземпляре.

Однако во многих сценариях восстановления база данных должна быть восстановлена на другой экземпляр SQL Server. Например, исходный экземпляр невозможно восстановить, или потому, что вы хотите переместить базу данных на другой сервер. В этом случае необходимо использовать следующую процедуру для восстановления зашифрованной базы данных:

1. Создайте главный ключ базы данных для базы данных master. Это не должен быть тот же главный ключ базы данных, который был использован в оригинальном экземпляре, но при восстановлении сервера после сбоя можно восстановить исходный главный ключ базы данных из резервной копии.

2. Создайте сертификат или ключ из резервной копии. Используйте инструкцию CREATE CERTIFICATE или CREATE ASYMMETRIC KEY для создания сертификата или ключа из резервной копии, созданной из оригинального ключа, используемого для шифрования базы данных. Новый сертификат или ключ должен иметь такое же имя, как и оригинал, и если использовали сертификат, необходимо восстановить открытый сертификат и закрытый ключ.

3. Восстановите базу данных. Теперь, когда ключ шифрования доступен на экземпляре SQL Server, можно восстановить базу данных как обычно.

В следующем примере кода показано, как восстановить зашифрованную резервную копию базы данных на новый экземпляр SQL Server:

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'Pa$$w0rd';
CREATE CERTIFICATE BackupCert
     FROM FILE = 'K:\Backups\Backup.cer'
     WITH PRIVATE KEY (
           DECRYPTION BY PASSWORD = 'CertPa$$w0rd',
           FILE = 'K:\Backups\Backup.key');
     GO
RESTORE DATABASE AdventureWorks
     FROM DISK = 'R:\Backups\AW_Encrypt.bak'
```

Восстановление отдельных страниц данных

В некоторых случаях в файлах базы данных могут быть повреждены отдельные страницы данных. В этом случае для восстановления БД можно восстановить отдельные страницы, что быстрее, чем восстановление всей базы данных или даже восстановление только поврежденного файла.

Существуют ограничения: могут быть восстановлены только страницы данных (восстановление страниц не может быть использовано ни для восстановления журнала транзакций, ни для восстановления страницы размещения, ни для восстановления загрузочных страниц базы данных или файлов). Восстановление страницы применяется к базам данных SQL Server, где используется модель полного восстановления или модель восстановления с неполным протоколированием, поддержка восстановления страниц осуществляется только для файловых групп, доступных для чтения и записи.

Первым признаком повреждения страниц, как правило, является возникновение ошибки 823 или 824 при запросе таблицы. Чтобы выявить потенциально поврежденные страницы, можно использовать команду DBCC CHECKDB и запрос к системной таблице *suspect_pages* в базе данных MSDB. Эта таблица содержит идентификатор каждой пораженной страницы, наряду с подробной информацией о потенциальной проблеме. В данной таблице приводится идентификатор страницы каждой страницы. С помощью этой информации, вы можете восстановить поврежденные страницы, используя инструкцию RESTORE DATABASE, или в среде SSMS с помощью диалогового окна Восстановить страницу (Restore Page).

Восстановление поврежденных страниц можно выполнить при сохранении базы данных в оперативном режиме. Online-восстановление поддерживается только в SQL Server Enterprise Edition. Можно выполнить восстановление страниц в автономном режиме. На время проведения восстановления страниц в автономном режиме база данных переключается в соответствующий режим. В конце последовательности восстановления база данных переходит в режим online.

Следующий пример кода выполняет онлайн-восстановление двух страниц:

```
-- Restore pages from the full backup
```

```
RESTORE DATABASE AdventureWorks PAGE='1:55, 1:207'
FROM DISK = 'R:\Backups\AdventureWorks.bak'
WITH FILE=1, NORECOVERY;
-- Apply the differential backup
RESTORE DATABASE AdventureWorks
FROM DISK = 'R:\Backups\AdventureWorks.bak'
WITH FILE=3, NORECOVERY;
-- Restore subsequent transaction log backups
RESTORE LOG AdventureWorks
FROM DISK = 'R:\Backups\AdventureWorks.bak'
WITH FILE=4, NORECOVERY;
-- Backup the log
BACKUP LOG AdventureWorks
TO DISK = 'R:\Backups\AW-Log.bak';
-- Restore the log to set the correct REDO LSN and recover
RESTORE LOG AdventureWorks
FROM DISK = 'R:\Backups\AW-Log.bak'
WITH RECOVERY;
```

Восстановление системных баз данных

Системные базы данных жизненно важны для работы экземпляра сервера. После каждого значительного обновления необходимо обязательно создавать резервные копии системных баз данных msdb, master и model. Процесс подробным объяснением шагов описаны в электронной документации [3].восстановления для системных баз данных не является идентичным. Каждая системная база данных имеет специфические требования восстановления, которые приведены в Таблице 7.

Таблица 7. Описание типов резервных копий в SQL Server

Системная БД	Описание
master	Надо создавать резервные копии: ДА Модель восстановления: Простая Восстановление: в однопользовательском режиме
model	Надо создавать резервные копии: ДА Модель восстановления: Настраивается пользователем Восстановление: с помощью флага трассировки -T3608
msdb	Надо создавать резервные копии: ДА Модель восстановления: Простая (по умолчанию) Восстановление: как любой пользовательской базы данных
tempdb /resource	Надо создавать резервные копии: НЕТ **tempdb** создается во время запуска экземпляра Восстановление **resource:** с помощью файла восстановления или установки

Как часто следует создавать резервные копии для системных баз данных?

Создавайте резервные копии базы данных master с такой частотой, которая необходима для адекватной защиты данных. Рекомендуется составить

расписание регулярного резервного копирования, которое можно дополнить созданием резервных копий после значительных обновлений.

Резервные копии базы данных model создаются только в том случае, если они необходимы для предприятия (например, сразу же после настройки параметров базы данных). Рекомендуется по мере необходимости создавать только полные резервные копии базы данных model. Поскольку база данных model невелика и редко изменяется, создавать резервную копию журнала не обязательно.

Резервную копию базы данных msdb надо создавать после каждого ее обновления.

Создать резервную копию системной базы данных tempdb нельзя.

База данных Resource находится в файле mssqlsystemresource.mdf, в котором содержится только код. Поэтому SQL Server не может создать резервную копию базы данных Resource. Исходя из того, что файл mssqlsystemresource.mdf является простым двоичным файлом (EXE), а не файлом базы данных, для создания его резервной копии можно выполнить простое резервное копирование файла или диска. Восстановить резервную копию файла mssqlsystemresource.mdf можно будет только вручную; при этом следует соблюдать осторожность, чтобы не перезаписать текущую базу данных Resource устаревшей или потенциально небезопасной версией.

Надо помнить, что системные базы данных могут быть восстановлены только из резервных копий, созданных той версией SQL Server, которая запущена на данном экземпляре сервера. Например, чтобы восстановить системную базу данных на экземпляре сервера, работающего под SQL Server 2012 с пакетом обновления SP1, необходимо использовать резервную копию базы данных, созданную после обновления экземпляра сервера до SQL Server 2012 с пакетом обновления SP1.

Для восстановления любой базы данных должен быть запущен экземпляр SQL Server. Для запуска экземпляра SQL Server необходимо, чтобы база данных master была доступна и хотя бы частично пригодна к использованию. Если база данных master непригодна к использованию, ее можно вернуть в нормальное состояние следующими способами:

- Восстановить базу данных master на основе актуальной резервной копии.
- Если экземпляр сервера удалось запустить, базу данных master можно восстановить из полной резервной копии.
- Перестроить базу данных master с нуля. При перестроении базы данных master все системные базы данных также перестраиваются

Подробная документация по этой важной теме в электронной документации Microsoft [1].

4.4 Восстановление на момент времени

Обычно хотят восстановить базу данных на самый последний момент времени – до точки сбоя. Хотя иногда требуется, чтобы восстановление базы данных было произведено ранее точки сбоя. Например, если транзакция ошибочно изменила какие-либо данные, может понадобиться восстановление базы данных на момент времени, предшествующий вводу неверных данных.

Для любого из этих вариантов, база данных должна использовать полную модель восстановления.

Можно в отдельных случаях использовать модель восстановления с неполным протоколированием. Однако если в модели восстановления с неполным протоколированием резервная копия журнала содержит изменения с неполным протоколированием, то в пределах этой резервной копии восстановление до момента времени невозможно. База данных должна быть восстановлена до конца резервной копии журнала транзакций.

Целевую точку восстановления можно указать как:

- определенный момент времени в пределах журнала транзакций;
- именованную метку, вставленную в запись журнала транзакций;
- регистрационный номер транзакции в журнале (номер LSN).

Для этого в операторе RESTORE DATABASE надо использовать один из параметров STOPAT, STOPATMARK или STOPBEFOREMARK.

Параметр STOPAT используется для восстановления на определенный момент времени. Аргумент в виде:

```
STOPAT = { 'datetime' | @datetime_var }
```

указывает, что база данных будет восстановлена в состояние, в котором она находилась на момент, соответствующий дате и времени, указанным или параметром datetime (обязательно тип данных smalldatetime или datetime) или @datetime_var. Если использована переменная, то она должна иметь тип данных varchar, char, smalldatetime или datetime. Только записи журнала транзакций, сделанные до указанных даты и времени, применяются к базе данных.

Вы можете не знать заранее, какой файл резервной копии журнала транзакций содержит транзакции на момент, когда восстановление должно произойти. Чтобы восстановить только изменения до определенного момента времени, для каждой команды восстановления журнала в последовательности RESTORE LOG укажите WITH STOPAT. Это гарантирует, что конечное время не будет пропущено, как показано в следующем примере:

```
RESTORE DATABASE database_name FROM full_backup
     WITH NORECOVERY;
RESTORE DATABASE database_name FROM differential_backup
     WITH NORECOVERY;
```

```
RESTORE LOG database_name FROM first_log_backup
     WITH STOPAT = time, RECOVERY;
… (additional log backups could be restored here)
RESTORE LOG database_name FROM final_log_backup
     WITH STOPAT = time, RECOVERY;
```

Обратите внимание, что параметр RECOVERY указан для каждого восстановления журнала транзакций, но фактически процесс восстановления не произойдет до момента времени, заданного параметром STOPAT, если он не находится в журнале транзакций. В резервной копии журнала транзакций должна присутствовать целевая точка восстановления:

- Если указанное время содержится в течение периода, охватываемого резервной копией журнала, команда RESTORE LOG … WITH STOPAT восстанавливает базу данных на это время.
- Если указанное время не содержится в течение периода, охватываемого резервной копией журнала транзакций, а оно
  - **ранее** первой содержащейся в резервной копии журнала транзакции, команда восстановления завершится неудачей и возвратит ошибку.
  - **позднее** последней содержащейся в резервной копии журнала транзакции, команда восстанавливает журналы, посылает предупреждающее сообщение, а база данных не восстанавливается, так что могут быть применены дополнительные резервные копии журнала транзакций.

Это гарантирует, что база данных будет восстановлена до желаемого момента, даже когда с параметром STOPAT указан параметр RECOVERY.

Параметр STOPATMARK задает восстановление до указанной точки восстановления.

```
STOPATMARK = { 'mark_name' | 'lsn:lsn_number' }
                [ AFTER 'datetime' ]
```

Инструкции RESTORE DATABASE и RESTORE LOG поддерживают параметр lsn_number. Этот параметр определяет регистрационный номер транзакции в журнале.

Параметр mark_name поддерживается только инструкцией RESTORE LOG. Этот параметр идентифицирует метку транзакции в резервной копии журнала. Т.е. при создании транзакции эта метка должна указываться, например:

```
BEGIN TRAN UpdPrc WITH MARK 'Start of night update process';
```

Если в инструкции RESTORE LOG опущен параметр AFTER datetime, то восстановление останавливается на первой метке с указанным именем. Если указан параметр AFTER datetime, то восстановление останавливается на первой метке с указанным именем непосредственно перед или после datetime. Если указанная отметка (номер LSN) или время назначены после создания последней резервной копии журналов, база данных остается в

невосстановленном состоянии, как если бы инструкция RESTORE LOG работала с параметром NORECOVERY.

Параметр STOPBEFOREMARK также задает восстановление до указанной точки восстановления.

```
STOPBEFOREMARK = { 'mark_name' | 'lsn:lsn_number' }
                 [ AFTER 'datetime' ]
```

Однако заданная параметром транзакция не включается в восстановление; после применения параметра WITH RECOVERY производится ее откат.

Если вы не знаете имя транзакции, которая была отмечена, можно запросить таблицу ***dbo.logmarkhistory*** *в базе данных* ***msdb***.

SQL Server Management Studio предоставляет графический пользовательский интерфейс (GUI), который позволяет легко восстановить базу данных на определенный момент времени.

Чтобы выполнить восстановление на момент времени с помощью среды SSMS, надо открыть диалоговое окно **Восстановление базы данных**. Для того, чтобы открыть диалоговое окно **Временная шкала резервного копирования**, надо щелкнуть **Временная шкала** (см. рисунок 5), где можно настроить определенную дату и время, когда необходимо остановить восстановление, используя либо окна **Данные** и **Время**, либо ползунок, чтобы указать конкретные дату и время, при которых процесс восстановления должен закончиться

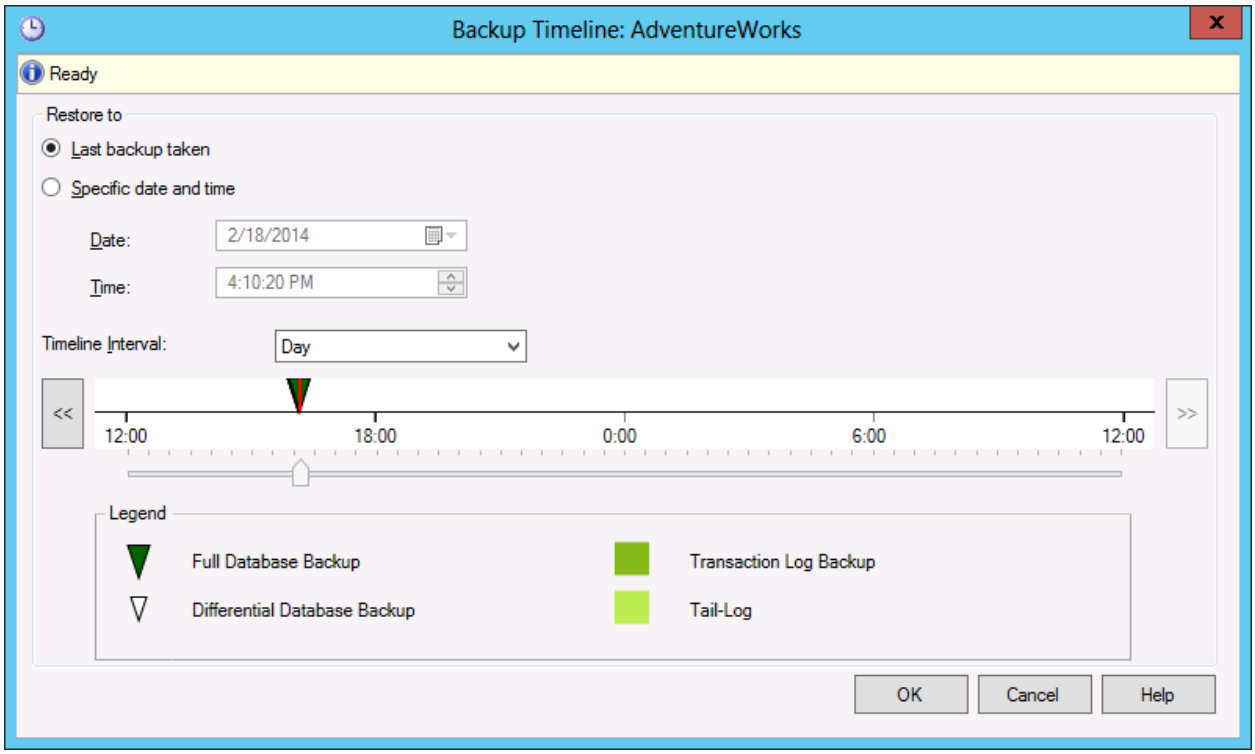

Рисунок 5 Окно диалога **Временная шкала резервного копирования**

5 ИМПОРТ И ЭКСПОРТ ДАННЫХ

Большое количество данных, находящихся в системе Microsoft SQL Server вводится непосредственно пользователями в приложениях, однако часто возникает необходимость импорта/экспорта данных. SQL Server предоставляет для этого набор инструментов: оператор BULK INSERT и функции OPENROWSET реализуются в ядре базы данных, утилиты BCP и служба SSIS являются внешними по отношению к ядру БД.

При выполнении массовых операций нередко возникает ряд проблем. Например, когда большие объемы данных должны быть вставлены в таблицы SQL Server, необходимо обеспечить наилучшую производительность. Для этого необходимо уметь правильно настраивать параметры для ограничений, триггеров и индексов. Не все данные могут быть введены пользователями базы данных построчно. Часто данные должны быть импортированы из внешних источников данных, таких как файлы или другие серверы баз данных. Кроме того, пользователи часто просят, чтобы данные из таблиц БД экспортировались в текстовые файлы. Может вызвать проблемы неправильно настроенные параметры сортировки. Корректировка параметров сортировки базы данных часто требует экспорта и повторного импорта данных из базы данных.

5.1 Обзор вопросов передачи данных

Передача данных. Хотя не все требования передачи данных идентичны, существует стандартный алгоритм, которого придерживается большинство задач передачи данных:

- Извлечение данных из источника данных.
- Некоторое преобразование данных, чтобы сделать их пригодными для целевой системы.
- Загрузка данных в целевую систему.

Вместе эти три шага обычно называются процесс ETL (Extract, Transform, Load), который может быть реализован с использованием ETL-инструментов.

В некоторых ситуациях более подходящим может быть процесс ELT (Extract, Load, Transform). Например, можно выполнять преобразования данных уже после загрузки данных в БД.

Извлечение данных. Извлечение данных, как правило, включает в себя выполнение запросов на источнике, или открытие и чтение исходных файлов, хотя есть и другие варианты.

В процессе извлечения данных преследуют две общие цели:

- Избежать чрезмерного воздействия на систему-источник. Например, не читают целые таблицы данных, когда нужно прочитать только

выбранные строки или столбцы. Кроме того, не перечитывают одни и те же данные и в любом случае избегают выполнения операторов, которые блокируют пользователей системы-источника.

- Обеспечить согласованность извлечения данных. Например, не включают в выходные данные одну строку из системы-источника больше, чем один раз.

Преобразование данных. Этап преобразования процесса ELT обычно включает в себя несколько таких шагов:

- Данные должны быть очищены. Например, может потребоваться удалить ошибочные данные или предоставить значения по умолчанию для пропущенных (отсутствующих) столбцов.
- Возможно, придется выполнять поиск. Например, входные данные могут включать имя клиента, но базе данных может понадобиться идентификатор клиента.
- Данные должны быть агрегированы. Например, входные данные могут включать в себя все транзакции, которые произошли за день, но базе данных могут понадобиться только ежедневные итоговые значения.
- Данные, возможно, потребуется разгруппировать. Это часто называют распределение данных. Например, входные данные могут включать в себя квартальные бюджеты, но база данных может потребовать ежедневный бюджет.

Помимо этих общих операций, данные, возможно, должны быть в некотором роде реорганизованы, например, сведение данных таким образом, чтобы столбцы становились строками, объединяющими нескольких столбцов источника в одну колонку, или разбиение одного исходного столбца на несколько столбцов.

Загрузка данных. После того, как данные преобразованы в соответствующий формат, их можно загрузить в целевую систему. Вместо выполнения операции вставки данных построчно, можно использовать специальные опции для массовой загрузки данных. Кроме того можно изменить временную конфигурацию для повышения производительности операции загрузки.

Средства для массового импорта и экспорта данных

SQL Server поддерживает массовый экспорт данных из таблиц SQL Server и массовый импорт данных в таблицы или несекционированные представления. Для выполнения этих задач SQL Server предоставляет набор инструментов. Важно понять, какой метод лучше всего использовать и для каких типов сценария. Обзор доступных средств приведен в Таблице 8.

Таблица 8. Основные средства для выполнения операций импорта/экспорта данных

Метод	Описание	Импорт данных	Экспорт данных
программа bcp,	Программа командной строки (Bcp.exe), массово экспортирующая и импортирующая данные и создающая файлы форматирования.	Да	Да
инструкция BULK INSERT	Инструкция T-SQL, импортирующая данные непосредственно из файла данных в таблицу базы данных или несекционированное представление.	Да	Нет
Инструкция INSERT … SELECT * FROM OPENROWSET(BULK…)	Инструкция Transact-SQL INSERT, использующая поставщик больших наборов строк OPENROWSET для массового импорта данных в таблицу. OPENROWSET – это табличная функция, которую можно использовать для подключения и извлечения данных из источников данных OLE DB. Полная информация о том, как подключиться к источнику данных, указывается в параметрах функции. SQL Server предлагает специальный поставщик OLE DB, именуемый BULK, который можно использовать с помощью функции OPENROWSET. Можно использовать OPENROWSET для подключения к СУБД другого типа.	Да	Нет
мастер импорта и экспорта	Мастер создает простые пакеты, которые импортируют и экспортируют данные в многочисленных распространенных форматах, включая базы данных, электронные таблицы и текстовые файлы.	Да	Да

Подробное описание использования этих средств и методов можно найти в электронной документации Microsoft [2].

Повышение производительности передачи данных

При выполнении операций массового экспорта/импорта для того, чтобы обеспечить наилучшую производительность, рекомендуется [2]:

- Минимизировать блокировки;
- Минимизировать ведение журнала транзакций;
- Отключить ограничения, индексы и триггеры.

Для чего надо минимизировать блокировки.

Блокировка используется в компоненте SQL Server Database Engine для синхронизации одновременного доступа нескольких пользователей к одному и тому же фрагменту данных. При изменении фрагмента данных транзакция удерживает блокировку, защищая изменения до конца транзакции. По умолчанию SQL Server управляет гранулярностью блокировок, начиная с уровня блокировки строк и пытаясь укрупнить блокировку, когда значительное число отдельных строк заблокированы в таблице. Управление большим количеством блокировок занимает ресурсы, которые могли бы использоваться для минимизации времени выполнения запросов. При выполнении операций массового импорта, как правило, нецелесообразно ставить блокировки на уровне строк, а заблокировать всю таблицу сразу.

Программа bcp и инструкция BULK INSERT и INSERT ... SELECT * FROM OPENROWSET(BULK...) позволяют указать, что на время выполнения операции массового импорта таблица будет блокирована (целиком). Если указана блокировка таблицы для операции массового импорта, то к таблице применяется блокировка массового обновления (BU) на время выполнения операции массового импорта. Блокировка BU позволяет одновременно выполнять массовый импорт данных в одну и ту же таблицу нескольким потокам и вместе с тем предотвращает доступ к таблице других процессов, не осуществляющих массовый импорт данных. Блокировка таблицы может повысить производительность массового импорта, снизив конкуренцию за эту таблицу.

В Таблице 9 перечислены квалификаторы, определяющие блокировку таблиц в командах массовой загрузки.

Таблица 9. Квалификаторы, задающие блокировку таблиц

Команда	**Квалификатор**	**Тип квалификатора**
bcp	-h " TABLOCK "	Подсказка
BULK INSERT	TABLOCK	Аргумент
INSERT ... SELECT * FROM OPENROWSET(BULK...)	WITH(TABLOCK)	Табличная подсказка

Почему целесообразно минимизировать ведение журнала транзакций.

В простой модели восстановления массовые операции производятся с минимальным протоколированием. В БД, использующей модель полного восстановления, все операции вставки строк полностью записываются в журнал транзакций. Во время импорта большого количества данных это может привести к быстрому заполнению журнала транзакций. Чтобы выполнять операции массового импорта с минимальным протоколированием в базе данных, которая обычно использует модель полного восстановления, ее рекомендуется сначала переключить на модель восстановления с неполным протоколированием, а после выполнения массового импорта данных снова переключить на модель полного восстановления.

Не все команды могут использовать минимальное протоколирование.

Способствовать минимизации ведения журнала помогут следующие рекомендации:

- Таблица не реплицируется.
- Указана блокировка таблицы (с помощью TABLOCK).
- Если у таблицы нет кластеризованного индекса, но имеется один или несколько некластеризованных индексов, страницы данных всегда протоколируются минимально. Однако, как идет запись страниц индекса в журнал, зависит от того, пуста ли таблица.
- Если таблица пуста, страницы индекса протоколируются минимально.
- Если таблица не пуста, страницы индекса протоколируются полностью.
- Если таблица имеет кластеризованный индекс и пуста, ведется минимальная запись страниц данных и индекса в журнал.
- Если таблица имеет кластеризованный индекс и не пуста, страницы данных и индекса протоколируются полностью, независимо от модели восстановления.

Это не исчерпывающий перечень ограничений, которые должны быть выполнены для того, чтобы минимизировать запись в журнал транзакций.

Отключение и перестроение индексов

Чтобы в процессе импорта или обновления не проверять каждое значение каждого индекса для каждой строки, можно повысить общую производительность путем отключения процесса обслуживания индексов до тех пор, пока все данные не будут загружены. Причем, для этого можно индекс не удалять из базы данных, а только отключить его, т.е. метаданные об индексе остаются, просто останавливается его обновление. Запросы не будут использовать отключённые индексы.

Отключение индекса:

- Предотвращает доступ пользователей к индексу;
- Предотвращает доступ к данным, если индекс кластеризованный;
- Сохраняет определение индекса в метаданных;
- Ускоряет импорт данных в таблицах.

Индекс можно отключить с помощью графического интерфейса в SSMS или с помощью инструкции ALTER INDEX. В следующем примере кода отключается индекс с именем idx_emailaddress в таблице БД dbo.Customer:

```
ALTER INDEX idx_emailaddress ON dbo.Customer
    DISABLE;
```

Можно отключить все индексы сразу в таблице БД dbo.Customer, как показано в следующем примере:

```
ALTER INDEX ALL ON dbo.Customer
    DISABLE;
```

Кластеризованный индекс определяет структуру таблицы. Если отключен кластеризованный индекс, таблица становится недоступной до тех пор, пока индекс не будет перестроен.

После завершения импорта данных можно перестроить (фактически создать заново) индексы для таблицы с помощью графических средств в среде SSMS, с помощью инструкции ALTER INDEX или команды DBCC DBREINDEX.

В следующем примере кода показано, как перестроить индекс с именем idx_emailaddress в таблице БД dbo.Customer:

```
ALTER INDEX idx_emailaddress ON dbo.Customer
     REBUILD;
```

Вы также можете использовать ключевое слово ALL с инструкции ALTER INDEX для перестроения всех индексов в указанной таблице (аналогично примеру отключения индекса).

Если загружен большой объем данных более эффективным может быть удаление существующих индексов и повторное создание индексов. Для повторного создания индекса, который заменит существующий, можно использовать инструкцию CREATE INDEX с параметром DROP_EXISTING, как показано в следующем примере:

```
CREATE INDEX idx_emailaddress ON dbo.Customer(EmailAddress)
     WITH (DROP_EXISTING = ON);
```

Отключение и включение ограничений

Ограничения используются, чтобы обеспечить соблюдение правил целостности данных.

Ограничения PRIMARY KEY и UNIQUE.

SQL Server создает индексы, чтобы обеспечить соблюдение этих ограничений. Чтобы отключить первичный ключ или ограничение уникальности, необходимо отключить индекс, связанный с ограничением. Это, как правило, используется только для некластеризованного первичного ключа. При повторном включении ограничения, соответствующие индексы автоматически восстанавливаются. Если во время перестройки индекса будут найдены повторяющиеся значения, повторное включение ограничения завершится ошибкой. По этой причине, если вы отключаете эти ограничения при импорте данных, то должны быть уверены, что импортируемые данные не будут нарушать соблюдение этих ограничений.

Ограничения FOREIGN KEY и CHECK.

Ограничения внешнего ключа используются, чтобы убедиться, что сущности в одной таблице, на которые ссылаются сущности из другой, на самом деле существуют. Например, поставщик должен существовать до того, как может быть введен заказ на поставку. Ограничения внешнего ключа при проверке ссылок используют первичный ключ или ограничения уникальности. Поэтому, если отключить первичный ключ или ограничение уникальности,

на которое оно указывает, то ограничение внешнего ключа автоматически отключается. Тем не менее, при повторном включении первичного ключа или ограничения уникальности, ограничения внешнего ключа, которые на них ссылаются, не будут включены автоматически.

Ограничения check можно использовать для ограничения значений, которые могут содержаться в столбце или взаимосвязи между значениями в нескольких столбцах таблицы.

Можно отключить и включить ограничения FOREIGN KEY и CHECK с помощью параметры CHECK и NOCHECK.

Пример кода для отключения и включения ограничения с именем SalaryCap:

```
ALTER TABLE Person.Salary NOCHECK CONSTRAINT SalaryCap;
ALTER TABLE Person.Salary CHECK SalaryCap;
```

Также можно отключить или включить все ограничения, заменив в инструкции ALTER TABLE имя ограничения на ключевое слово ALL.

5.2 Копирование и перемещение Баз данных

В некоторых случаях необходимо скопировать или переместить всю базу данных с одного экземпляра SQL Server на другой. Это можно выполнить одним из следующих способов:

- с помощью мастера копирования баз данных (Copy Database Wizard);
- при помощи отсоединения (detach) и присоединения (attach);
- путем создания (backup) и восстановления (restore) резервных копий;
- приложений уровня данных (Data-tier applications).

Мастер копирования баз данных

С помощью мастера копирования баз данных можно легко перемещать или копировать базы данных и их объекты с одного сервера на другой, без перерывов в работе сервера. Можно также обновить базы данных с прошлой версии SQL Server до версии SQL Server 2014. С помощью этого мастера можно сделать следующее.

- Выбрать исходный и целевой серверы.
- Выбрать базы данных для перемещения, копирования или обновления.
- Указать расположение файлов для баз данных.
- Создать имена входа для целевого сервера.
- Копировать дополнительные вспомогательные объекты, задания, пользовательские хранимые процедуры и сообщения об ошибках.
- Задать расписание перемещения или копирования баз данных.

Мастер предоставляет два метода копирования или перемещения базы данных. Он может быть настроен на использование метода отсоединения и присоединения: это самый быстрый вариант, но имеет недостаток, так как

исходная база данных должна находиться в автономном режиме. Второй – метод SMO, использует библиотеки объектов управления SQL Server (Server Management Objects) для создания объектов и передачи данных. Этот метод выполняет чтение определения каждого объекта базы данных-источника и создает каждый из этих объектов в целевой базе данных. После этого происходит перенос данных из исходных таблиц в целевые таблицы с воссозданием индексов и метаданных. Этот вариант медленнее, но исходная база данных во время копирования может находиться в режиме online.

Если в мастере выбран параметр «Переместить», то после перемещения базы данных мастер автоматически удаляет базу данных-источник. Если выбран параметр «Копировать», исходная база данных остается без изменений.

Существует ряд ограничений:

- Мастер копирования баз данных недоступен в выпуске Express;
- Мастер копирования базы данных нельзя использовать для перемещения или копирования системных баз данных;
- После обновления базы данных возврат к предыдущей версии невозможен;
- На сервере назначения должен быть запущен агент SQL Server.

Если исходная база данных используется, когда мастер пытается переместить или скопировать базу данных, операция не выполняется.
Запуск мастера копирования баз данных требует привилегий системного администратора (sysadmin) на обоих экземплярах (как на исходном, так и на целевом сервере) и наличия сетевого подключения.

Использование операций Detach и Attach

Отсоединение базы данных в SQL Server 2014 можно выполнить с помощью среды SSMS или Transact-SQL. Отсоединенные файлы останутся на диске и могут быть повторно присоединены с помощью среды SSMS или с помощью инструкции CREATE DATABASE с параметрами FOR ATTACH или FOR ATTACH_REBUILD_LOG. Файлы можно также переместить на другой сервер и подсоединить там.

Не рекомендуется подключать или восстанавливать базы данных, полученные из неизвестных или ненадежных источников. В этих базах данных может содержаться вредоносный код, вызывающий выполнение непредусмотренных инструкций Transact-SQL или появление ошибок из-за изменения схемы или физической структуры базы данных. Перед тем как использовать базу данных, полученную из неизвестного или ненадежного источника, выполните на тестовом сервере инструкцию DBCC CHECKDB для этой базы данных, а также изучите исходный код в базе данных, например, хранимые процедуры и другой пользовательский код.

Для отсоединения БД используется системная хранимая процедура sp_detach_db, которой надо передать как минимум один обязательный параметр – имя отсоединяемой базы данных.

Следующий код выполняет отсоединение БД AdventureWorks2012:

```
EXEC sp_detach_db 'AdventureWorks2012', 'true';
```

В этом примере указан второй – необязательный – параметр skipchecks в значении TRUE, чтобы не выполнять инструкцию UPDATE STATISTICS. Чтобы явно запустить инструкцию UPDATE STATISTICS, надо указать значение FALSE.

При выполнении операции отсоединения, необходимо учитывать следующие обстоятельства:

- Невозможно отсоединить базу данных, если она в настоящий момент используется. Для отсоединения базы данных требуется монопольный доступ, для чего надо переключить ее в режим SINGLE_USER.

  Например, следующая инструкция ALTER DATABASE получает монопольный доступ к базе данных AdventureWorks2012 после отключения от этой базы данных всех текущих пользователей:

  ```
  USE master;
  ALTER DATABASE AdventureWorks2012
   SET SINGLE_USER;
  GO
  ```

- База данных помечена как подозрительная. Подозрительную базу данных перед ее отсоединением необходимо перевести в аварийный режим.
- База данных является системной базой данных.
- Перед отсоединением базы данных необходимо удалить все моментальные снимки, если таковые имеются.
- Для выполнения операций требуется членство в предопределенной роли сервера sysadmin.
- При отсоединении базы данных все метаданные удаляются. Если эта база данных была базой данных по умолчанию для учетной записи входа, базой данных по умолчанию становится **master**.

Копирования базы данных с помощью Backup и Restore

В SQL Server 2014 можно создать новую базу данных, восстановив резервную копию пользовательской базы данных, созданной в SQL Server 2005 или более поздней версии. Однако резервные копии баз данных master, model и msdb, созданных в более ранней версии SQL Server, восстановить на SQL Server 2014 невозможно. Кроме того, резервные копии, созданные в SQL Server 2014, невозможно восстановить в более ранних версиях SQL Server.

При использовании резервного копирования и восстановления для копирования базы данных на другой экземпляр SQL Server компьютер-

источник и целевой компьютер могут быть любой платформой, на которой запускается SQL Server.

Основные этапы:

1. Создайте резервную копию базы данных источника, которая может находиться на экземпляре SQL Server 2005 или более поздней версии. Компьютер, на котором запущен этот экземпляр SQL Server, называется компьютером-источником.
2. На компьютере, куда нужно скопировать базу данных (целевой компьютер), подключите экземпляр SQL Server, на котором будет восстановлена база данных. При необходимости создайте те же устройства резервного копирования на целевом экземпляре сервера, что использовались для резервного копирования баз данных-источников.
3. Восстановите резервную копию базы данных источника на целевом компьютере. При восстановлении базы данных автоматически создаются все ее файлы.

Рассмотрим дополнительные вопросы, которые могут повлиять на процесс копирования БД.

При восстановлении базы данных необходимые файлы базы данных создаются автоматически. По умолчанию у файлов, созданных SQL Server в процессе восстановления, те же имена и пути, что и у файлов резервной копии исходной базы данных на компьютере-источнике. В некоторых ситуациях при восстановлении базы данных необходимо указать сопоставление дисков, имена файлов или путь для восстановления. Например, на целевом диске может быть недостаточно свободного места, или используется имя базы данных, которое уже существует на целевом сервере восстановления, а имя каждого из ее файлов совпадает с именем файла базы данных в резервном наборе данных.

Во избежание ошибок и непредвиденных последствий перед операцией восстановления можно использовать таблицы журнала backupfile, чтобы найти в резервной копии файлы базы данных и журнала, которые планируется восстановить.

При восстановлении базы данных на другом компьютере имя входа пользователя SQL Server или Microsoft Windows, начавшего процесс восстановления, автоматически становится именем владельца базы данных. При восстановлении базы данных системный администратор или владелец новой базы данных могут сменить ее владельца. Для предотвращения несанкционированного восстановления базы данных устанавливайте пароли на носители или сами резервные копии.

Чтобы обеспечить целостность работы пользователей и приложений при восстановлении базы данных на другой экземпляр сервера, на новом экземпляре необходимо повторно создать некоторые или все метаданные, например имена входа и задания [2].

6 УПРАВЛЕНИЕ БЕЗОПАСНОСТЬЮ SQL SERVER

Надлежащая защита данных имеет жизненно важное значение в любом приложении, и понимание того, как обеспечивать безопасность на уровне сервера и базы данных является ключевым требованием для администратора базы данных.

6.1 Введение в безопасность SQL Server

SQL Server предназначен для защиты данных платформы и включает ряд функций безопасности. Архитектура безопасности в SQL Server основана на прочно установившихся принципах, с которыми необходимо ознакомиться перед настройкой отдельных параметров безопасности.

Общая концепция безопасности

Безопасность является одной из основных функций всех систем корпоративного программного обеспечения, поэтому многие из понятий, относящихся к безопасности, схожи в различных системах.

На рисунке 6 представлена базовая концепция безопасности. Она включает:

- Защищаемые объекты (Securables);
- Субъектов или Участников (Principals);
- Разрешения (Permissions).

Рисунок 6 Общая концепция безопасности

Защищаемые объекты в SQL Server

К защищаемым объектами относятся ресурсы, доступ к которым регулируется системой авторизации компонента Database Engine. Некоторые защищаемые объекты могут храниться внутри других защищаемых объектов, создавая иерархии «областей», которые сами могут защищаться. К областям защищаемых объектов относятся сервер, база данных и схема. Иерархическая структура защищаемых объектов представлена на рисунке 7.
В SQL Server защищаемые объекты на верхнем уровне экземпляра называются объектами уровня сервера. К ним относятся:

- ✓ Имя входа (Logins)
- ✓ Роль сервера (Server roles)
- ✓ база данных (database)
- ✓ Конечная точка (Endpoints)
- ✓ Группа доступности (Credentials)

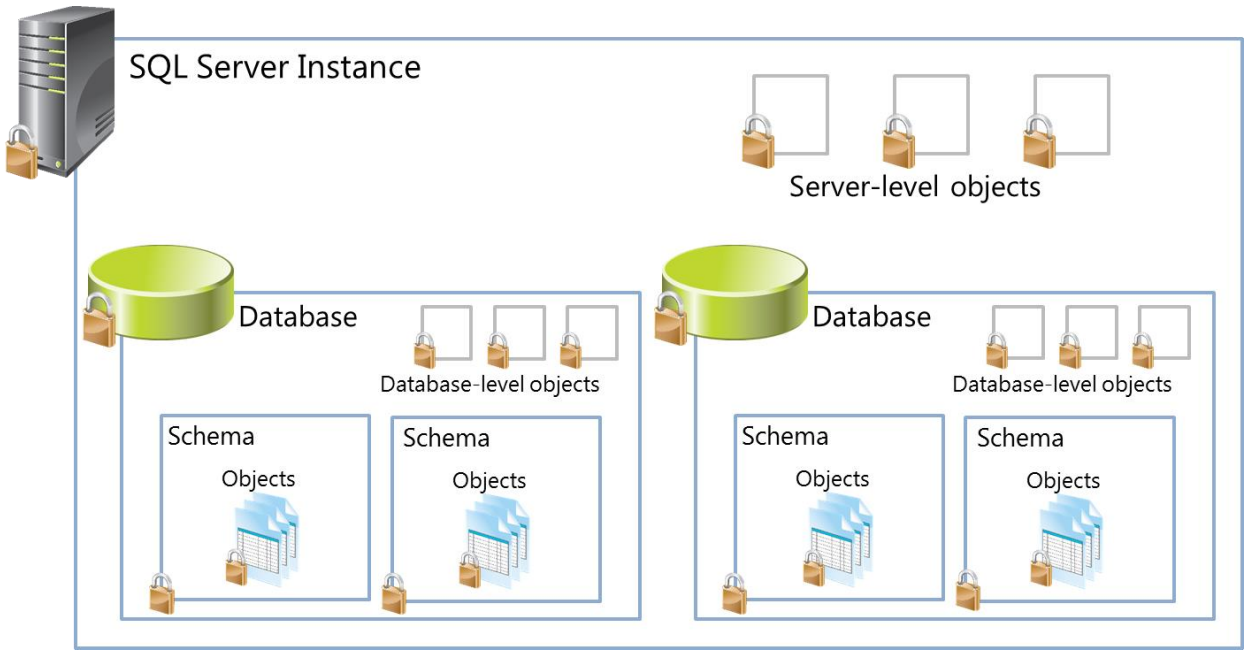

Рисунок 7 Защищаемые объекты на экземпляре SQL Server

Защищаемые объекты уровня базы данных:

- ✓ Схема
- ✓ Роль приложения
- ✓ Сборка
- ✓ Асимметричный ключ
- ✓ Сертификат
- ✓ Контракт
- ✓ Полнотекстовый каталог
- ✓ Привязка удаленной службы
- ✓ Тип сообщений;
- ✓ Роль (база данных)
- ✓ Маршрут
- ✓ Поиск в списке свойств
- ✓ Служба
- ✓ Полнотекстовый список стоп-слов
- ✓ Симметричный ключ
- ✓ Пользователь

Защищаемые объекты уровня схемы:

- ✓ Тип
- ✓ Коллекция схем XML
- ✓ Объект:
  - ➢ Таблица
  - ➢ Представление
  - ➢ Процедура
  - ➢ Функция
  - ➢ Статистика
  - ➢ Синоним
  - ➢ Очередь

По определению и клиент, и сервер базы данных являются защищаемыми субъектами безопасности. Данные сущности могут пройти взаимную проверку подлинности перед установкой безопасного сетевого соединения.

Некоторые защищаемые объекты являются также участниками. Например, имя входа является участником, который обеспечивает доступ к экземпляру SQL Server; но это также и защищаемый объект, потому что на нем могут быть выполнены действия (например, отключение или удаление), которые требуют разрешения.

Участники в SQL Server

Применительно к компоненту Database Engine *Участники* (или *Субъекты*) – это сущности, которые могут запрашивать ресурсы SQL Server. Наиболее часто в роли участников выступают имена входа и пользователи базы данных. Как и другие компоненты модели авторизации SQL Server, участников можно иерархически упорядочить. В Таблице 10 перечислены участники безопасности, а иерархия субъектов представлена на рисунке 8.

Таблица 10. Структура участников SQL Server по области определения

Иерархия	**Участники**
уровня Windows	• Имя входа домена Windows • Локальное имя входа Windows
уровня сервера	• Имя входа SQL Server • Роль сервера
уровня базы данных	• Пользователь базы данных • Роль базы данных • Роль приложения

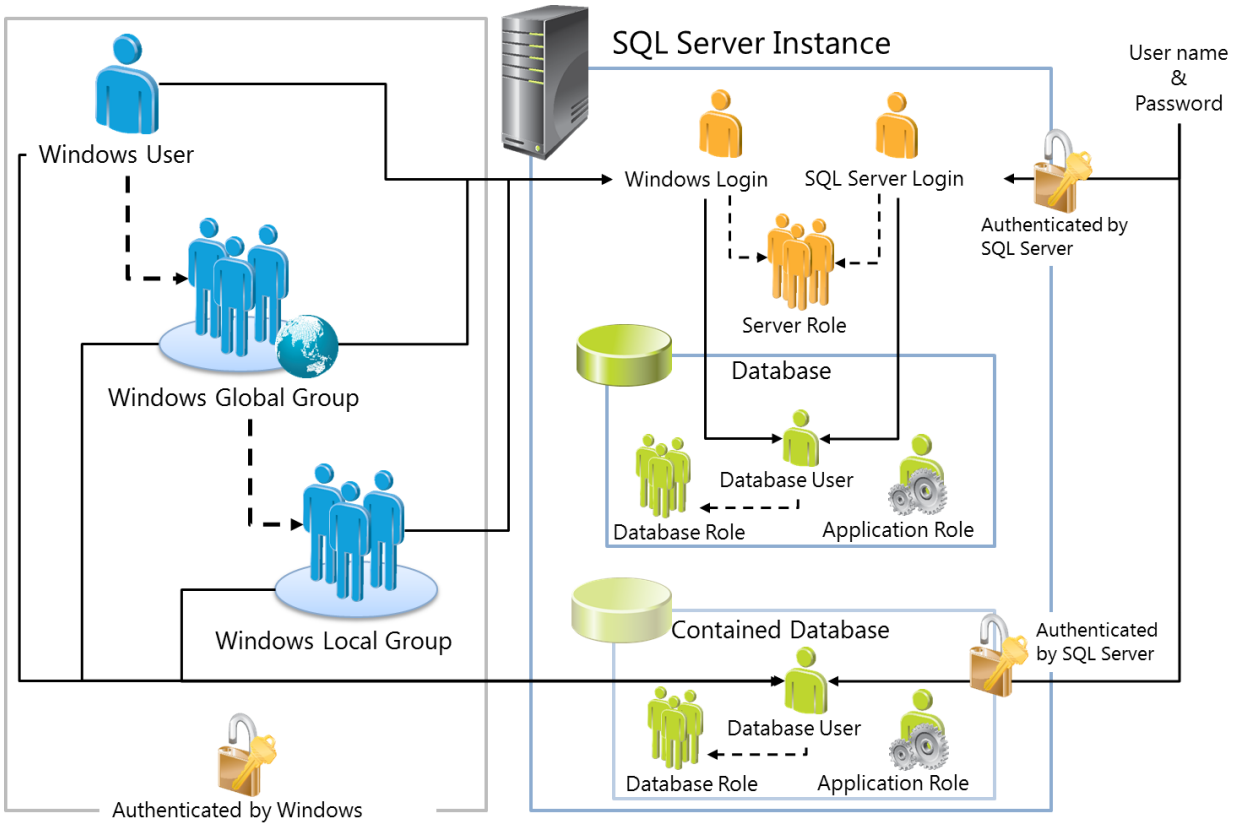

Рисунок 8 Субъекты безопасности на экземпляре SQL Server

Область влияния субъекта зависит не только от области определения (Windows, сервер, база данных), но и от того, неделимый это субъект или коллективный. Имя входа Windows является примером индивидуального (неделимого) субъекта, а группа Windows — коллективного. Каждый субъект имеет идентификатор безопасности (SID).

Имя входа – это субъект безопасности, с помощью которого система безопасности может проверить подлинность лица или сущности. Имя входа необходимо пользователю для соединения с SQL Server.

SQL Server поддерживает два типа имен входа:

- Имена входа Windows. Это учетные записи безопасности, управляемые с помощью Windows, например, пользователь или группа Windows. SQL Server не проверяет подлинность этих имен входа, а скорее доверяет Windows проверку их личности. По этой причине подключения к SQL Server с помощью имени входа Windows часто называют доверенные соединения.
- Имена входа SQL Server. Это имена входа с учетными данными безопасности, которые определены в базе данных ***master***. SQL Server проверяет подлинность этих имен входа путем проверки пароля. Существуют только для обратной совместимости с приложениями и пользователями, которым необходимо получать доступ к SQL Server, используя явную учетную запись пользователя и пароль.

При использовании имен входа Windows важно отметить, что имя входа Windows в SQL Server может ссылаться на отдельного пользователя Windows, на глобальную группу домена, определенную в Active Directory или на локальную группу (локальная группа домена в Active Directory или локальная группа Windows Server, на котором находится SQL Server). Имя входа Windows, которое ссылается на группу, позволяет всем пользователям в этой группе получить доступ к экземпляру SQL Server.

Использование имен входа групп Windows может значительно упростить администрирование SQL Server. Пользователи Windows добавляются к глобальным группам в зависимости от их роли в организации, а глобальные группы добавляются в локальные группы на основе конкретных требований доступа к SQL Server. По мере появления новых пользователей, изменения статуса существующих пользователей или их ухода, доступ к SQL Server контролируется с помощью членства в группе Active Directory без необходимости внесения каких-либо изменений в SQL Server. Следует также отметить, что доступ, основанный только на именах входа групп Windows, может сделать более сложным решение вопросов тестирования и устранения неполадок в рамках SQL Server; но в целом можно на это пойти, чтобы получить долгосрочные преимущества.

Роли сервера являются участниками безопасности, к которым можно добавлять имена входа, чтобы упростить управление разрешениями. SQL Server 2014 поддерживает два типа ролей:

- Фиксированные роли уровня сервера: системные (предопределенные) роли, которые автоматически получают необходимые разрешения для выполнения конкретных задач управления на уровне сервера.
- Роли сервера, определяемые пользователем: роли, которые администраторы могут создать для того, чтобы определить пользовательские группы управления на уровне сервера.

Участники уровня базы данных.

Наличие доступа к серверу (само по себе) не означает, что имя входа имеет доступ к пользовательским базам данных на сервере.

Чтобы логин имел доступ к базам данных, надо сопоставить имя входа (*login*) и пользователя базы данных (*database user*) в базе данных. Можно добавить пользователей базы данных к роли уровня базы данных (*database-level roles*) для упрощения управления разрешениями.

Пользователь базы данных – участник уровня базы данных, который обычно сопоставляется с именем входа на уровне сервера. Пользователи базы данных часто имеют одинаковые имена с именами входа, с которыми сопоставлены, но это необязательно. Даже можно сопоставить имена входа с разными именами пользователей в каждой базе данных.

Наилучшим способом считается сопоставление имен входа именам пользователей базы данных.

Существует одно исключение из этого правила: если база данных была настроена в качестве автономной базы данных. Автономные БД изолированы от других баз данных и от экземпляра SQL Server, на котором они размещены. Полностью автономная база данных содержит все параметры и метаданные, необходимые для определения базы данных, а ее конфигурация не зависит от Database Engine. Поэтому пользователи автономной базы данных не сопоставляются с именами входа. Можно пользователей автономной БД сопоставить с учетной записью Windows, или настроить на проверку подлинности SQL Server на уровне базы данных.

Можно добавить пользователей базы данных к *ролям базы данных* для упрощения управления разрешениями. SQL Server поддерживает:

- Фиксированные роли уровня базы данных: системные роли, которые инкапсулируют разрешения, необходимые для выполнения общих задач.
- Роли базы данных, определяемые пользователем: пользовательские роли, которые администраторы могут создать для группы пользователей с аналогичными требованиями к доступу.

В среде, где все имена входа основаны на группах Windows, пользователи базы данных, основанные на этих учетных записях, ведут себя во многом так же, как роли, так что вы можете предоставлять разрешения для базы данных непосредственно пользователям групп Windows вместо того, чтобы создавать пользовательские роли. Тем не менее, создание пользовательских ролей БД может быть полезно для объединения пользователей с аналогичными требованиями доступа при использовании имен входа Windows и имен входа SQL Server.

Роль приложения – участник уровня базы данных, позволяющий приложению выполняться в пределах базы данных со своими, подобными пользовательским, правами доступа. Когда роль приложения активна, SQL

Server обеспечивает соблюдение прав доступа, которые применяются к роли приложения, а не пользователя базы данных.

Роли приложений не содержат элементов и по умолчанию находятся в неактивном состоянии. Роли приложений работают с обоими режимами проверки подлинности. Роли приложений активируются с помощью процедуры sp_setapprole, которая требует указания пароля. Так как роли приложений являются участниками на уровне базы данных, они имеют доступ к другим базам данных только с разрешениями, предоставленными учетной записи пользователя guest в этих базах данных. Таким образом, любая база данных, в которой была отключена учетная запись пользователя guest , не будет доступна для ролей приложений в других базах данных.

Разрешения в SQL Server

Управление доступом к защищаемым объектам осуществляется путем предоставления или отзыва разрешений, либо путем добавления имен входа и пользователей к ролям, имеющим доступ.

Архитектуры безопасности часто являются иерархическими, в первую очередь для того, чтобы упростить управление разрешениями. В архитектуре иерархической безопасности защищаемые объекты могут содержать другие защищаемые объекты, и участники могут содержать другие сущности (Principal), например, пользователи могут быть добавлены в группу. Обычно разрешения наследуются как в иерархии защищаемых объектов, так и иерархиями участников. Для тонкой настройки доступа можно явно переопределить унаследованные разрешения на разных уровнях иерархии.

Такая иерархическая организация упрощает управление разрешениями:

- Меньшее количество индивидуальных разрешений должны быть предоставлены, что снижает риск неправильной конфигурации безопасности. Можно задать общие разрешения, которые требуются на самом высоком уровне в иерархии и только применять явные переопределения разрешения далее вниз по иерархии для обработки исключительных случаев.
- После того, как были установлены разрешения, ими можно управлять с помощью членства в группе. Это облегчает управление разрешениями в средах, где приходят новые пользователи, а существующие пользователи уходят или меняют роль.

При планировании конфигурации безопасности надо учитывать следующие рекомендации:

- Предоставляйте каждому участнику только те разрешения, которые ему на самом деле нужны.
- Используйте наследование прав доступа, чтобы минимизировать количество неявных разрешений, которые должны быть установлены для того, чтобы обеспечить необходимый уровень доступа.
- Используйте основные контейнеры, такие как группы или роли, чтобы создать уровень абстракции между участниками и разрешениями на

доступ к защищаемым объектам. Затем используйте членство этих групп для управления доступом к ресурсам с помощью разрешений, которые вы определили. Кадровые изменения никогда не должны приводить к корректировке разрешений.

Оператор GRANT.

Предоставляет разрешения на выполнение действий на защищаемом объекте участникам. Субъект, который не получил разрешение, не может выполнить действие, связанное с разрешением.

Общая структура оператора может быть условно представлена, как:

GRANT < permissions > ON < securables > TO < principals >

Полное описание синтаксиса инструкции GRANT очень сложно, т.к. зависит от многих факторов: на какие защищаемые объекты, какие разрешения, кому даются. Например, защищаемыми объектами могут быть и OBJECT, и LOGIN, и DATABASE, и ROLE, и SCHEMA, и USER, причем состав участников, которым можно предоставлять разрешения, меняется в зависимости от защищаемого объекта. От защищаемых объектов зависят и допустимые разрешения. В одном операторе GRANT можно предоставить сразу несколько разрешений нескольким участникам на несколько объектов.

Можно использовать параметр ALL:

GRANT ALL ON < securables > TO < principals >

Этот параметр устарел и сохранен только для поддержки обратной совместимости. Но надо учитывать, что он не предоставляет все возможные разрешения: например, если защищаемым объектом является база данных, аргумент ALL относится только к разрешениям BACKUP DATABASE, BACKUP LOG, CREATE DATABASE, CREATE DEFAULT, CREATE FUNCTION, CREATE PROCEDURE, CREATE RULE, CREATE TABLE и CREATE VIEW.

Разрешения могут быть предоставлены явным образом или могут быть унаследованы. Наследование разрешений применяется к основным иерархиям (например, если роль базы данных получает разрешение SELECT на схему, всем пользователям БД, которые являются членами этой роли, неявно предоставляется разрешение SELECT на эту схему). Также наследование разрешений применяется к защищаемым иерархиям (например, роль базы данных, которой было предоставлено разрешение SELECT на уровне схемы, неявно получает разрешение SELECT и на все объекты внутри схемы).

Унаследованные разрешения являются накопительными. Например, если роли базы данных было предоставлено разрешение SELECT на схему, и пользователю, который является членом этой роли, явным образом было предоставлено разрешение UPDATE для таблицы в схеме, то пользователь

имеет и разрешение SELECT на таблицу (наследуется через членство в роли), и разрешение UPDATE (непосредственно предоставленное пользователю).

Если оператор GRANT используется с параметром WITH GRANT OPTION, то получающему разрешение субъекту безопасности будет дана возможность предоставлять указанное разрешение другим участникам (учетным записям безопасности). Использовать эту опцию можно только применительно к объектным привилегиям и с большой осторожностью, т.к. можно потерять контроль над безопасностью защищаемого объекта.

Пример 1: пользователь БД *WilJo* получает разрешение SELECT на схему *Person* с правом предоставлять это разрешение другим пользователям БД:

```
GRANT SELECT ON SCHEMA :: Person TO WilJo
     WITH GRANT OPTION;
```

Пример 2: роли предоставляется разрешение на выполнение процедуры (оператор выполняет database owner):

```
GRANT EXECUTE ON TestProc TO TesterRole
     WITH GRANT OPTION;
```

К роли добавляем еще одного пользователя *User1*:

```
EXEC sp_addrolemember TesterRole, User1;
```

Если новый пользователь попытается выполнить следующий оператор, то получит ошибку:

```
GRANT EXECUTE ON TestProc TO User2;
```

Это объясняется тем, что пользователь *User1* не имеет разрешения на предоставление разрешений. Но оператор:

```
GRANT EXECUTE ON TestProc TO User2 AS TesterRole;
```

будет выполнен успешно, потому что *User1* выполняет эту инструкцию, как член роли *TesterRole*.

Подробную информацию надо смотреть в документации [1-3].

Оператор DENY.

С помощью инструкции DENY может быть сделано исключение к совокупным наследуемым разрешениям. Инструкция DENY явно запрещает конкретное разрешение участника на защищаемый объект и отменяет любые другие явные и наследуемые разрешения, которые могут быть предоставлены участнику.

Формат инструкции DENY аналогичен формату инструкции GRANT, как показано в следующем псевдокоде:

DENY < permissions > ON < securables > TO < principals >

Фактически это означает «ЗАПРЕТИТЬ (Отказать) субъектам безопасности в выдаче разрешений на защищаемые объекты».

Например, пользователь, который является членом роли базы данных, которая в свою очередь имеет разрешение SELECT на схему, автоматически имеет разрешение SELECT на все таблицы и представления в этой схеме. Если схема содержит таблицу, к которой пользователь не должен иметь

доступ, можно запретить (забрать) разрешение SELECT для пользователя. В этом случае, даже если пользователь унаследовал разрешение SELECT через членство в роли базы данных (которая в свою очередь унаследовали разрешение SELECT от родительской схемы), он не сможет запросить таблицу.

Обратите внимание, что разрешения DENY наследуются и не могут быть переопределены вниз по иерархии. Например, если забрать (DENY) у пользователя разрешение SELECT на схему, предоставление (GRANT) разрешения SELECT на отдельную таблицу в схеме не позволит пользователю обратиться к этой таблице. Важно помнить, что разрешение DENY не может использоваться для предотвращения доступа к защищаемому объекту его владельца или члена серверной роли sysadmin.

DENY следует использовать осторожно и нечасто. Необходимость запрещать много разрешений, как правило, указывает на возможную проблему в схеме безопасности.

Оператор REVOKE.

Чтобы удалить ранее предоставленные или запрещенные разрешения, можно использовать инструкцию REVOKE, как показано в следующем псевдокоде:

REVOKE < permissions > ON < securables > FROM < principals >

Обратите внимание, что оператор REVOKE удаляет только явные разрешения, его нельзя использовать для переопределения наследуемых разрешений.

Если выданное разрешение включало опцию WITH GRANT OPTION, можно использовать оператор REVOKE GRANT OPTION FOR, чтобы без отзыва самого разрешения отменить возможность предоставлять это разрешение другим. Совместно с параметром GRANT OPTION FOR можно использовать предложение CASCADE для отзыва разрешения у других лиц, которым уже было предоставлено разрешение указанным участником [2].

Распространенная ошибка – попытка удалить GRANT с помощью команды DENY вместо REVOKE. Это может повлечь за собой проблемы, если пользователь получает разрешения из нескольких источников, что часто встречается.

Продемонстрируем этот принцип на следующем примере:

Роль *Sales* получает разрешения SELECT для доступа к таблице *OrderStatus*:

```
GRANT SELECT ON OBJECT::OrderStatus TO Sales;
```

Ted является членом роли *Sales*. Кроме того, ему предоставлено разрешение по его собственному имени пользователя посредством инструкции:

```
GRANT SELECT ON OBJECT::OrderStatus TO Ted;
```

Предположим, администратор хочет удалить разрешение GRANT для роли *Sales*.

- Если администратор (правильно) выполняет инструкцию:

```
REVOKE SELECT ON OBJECT::OrderStatus TO Sales;
```

то пользователь *Ted* не утрачивает разрешение SELECT к таблице *OrderStatus* по своей собственной инструкции GRANT.

- Если администратор (неправильно) выполняет инструкцию:

```
DENY SELECT ON OBJECT::OrderStatus TO Sales;
```

то пользователь *Ted*, как член роли *Sales*, утрачивает разрешение SELECT, так как команда DENY для роли *Sales* переопределяет его собственную инструкцию GRANT.

Действующие разрешения.

Действующие разрешения для участника на доступ к защищаемому объекту соблюдаются SQL сервером на основе:

- Явных разрешений, определенных для этого участника для данного объекта;
- Разрешений, унаследованных участником через роль или членство в группах;
- Разрешений, унаследованных от предков защищаемого объекта.

Действующие разрешения для имен входа SQL Server и отдельных логинов Windows можно просмотреть двумя способами:

- В среде SSMS просмотрите вкладку **Разрешения** диалогового окна свойств защищаемого объекта или вкладку **Защищаемые объекты** диалогового окна свойств участника. Здесь вы можете выбрать сочетание участника и защищаемого объекта, которое хотите просмотреть или отметить (галочкой) на вкладке **Действующие** панели разрешений.
- Используя инструкцию EXECUTE AS для переключения контекста выполнения сеанса на заданное имя входа (на уровне сервера) или имя пользователя (на уровне базы данных), а затем запрос к системной функции sys.fn_my_permissions, указав защищаемый объект, для которого требуется просмотреть действующие разрешения.

В следующем примере кода показано, как выполнить просмотр действующих разрешений для имени входа *AdventureWorks\RosieReeves* на таблицу *dbo.Products*:

```
EXECUTE AS LOGIN = 'AdventureWorks \ RosieReeves'
SELECT *
   FROM sys.fn_my_permissions ( 'dbo.Products' 'Object'.);
REVERT
```

Инструкция REVERT переключает (возвращает) контекст выполнения в контекст участника, вызывавшего последнюю инструкцию EXECUTE AS.

Имена входа групп Windows и пользователей, основанных на группах Windows, нельзя использовать в инструкции EXECUTE AS (как нельзя использовать роль, сертификат, ключ или встроенную учетную запись). Таким образом, ни один из этих методов нельзя использовать для просмотра действующих разрешений для имени входа на основе группы Windows. Каждый пользователь Windows может войти и запросить системную функцию sys.fn_my_permissions для себя, но так как пользователи Windows

могут быть добавлены к более чем одной группе Windows, результаты для каждого пользователя могут различаться в зависимости от того, к каким группам они принадлежит.

6.2 Управление безопасностью уровня сервера

Реализация безопасности для SQL Server обычно начинается на уровне сервера, где пользователи проходят проверку подлинности на основе имен входа и организованы в роли уровня сервера, чтобы упростить управление разрешениями.

Модели безопасности приложений

Необходимо убедиться, что доступ к экземпляру SQL Server могут получить только пользователи, прошедшие проверку подлинности. Однако прежде чем приступить к созданию имен входа, необходимо обдумать архитектуру безопасности приложений, которую должна поддерживать база данных.

Модель безопасности доверенного сервера приложений, представленная на рисунке 9, обычно используется в крупномасштабных корпоративных приложениях, веб-сайтах и Интернет-сервисах.

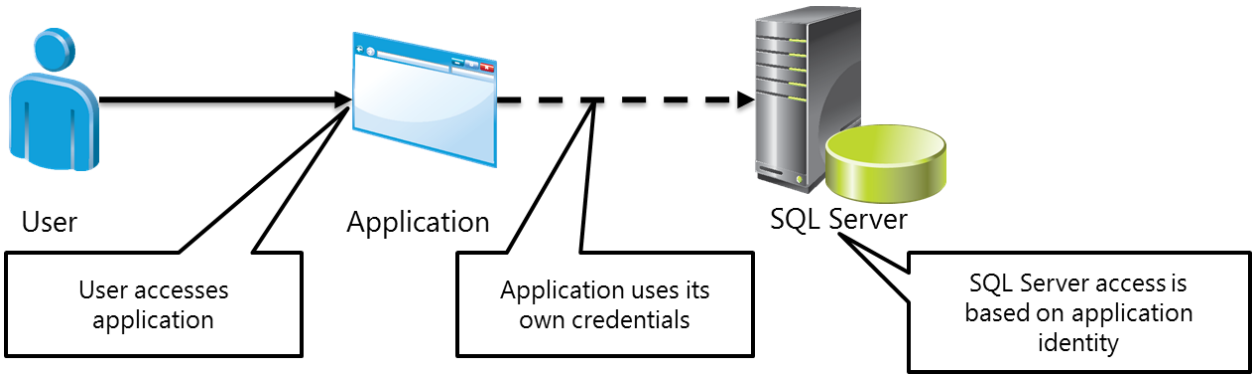

Рисунок 9 Модель безопасности доверенного сервера приложений

В этой модели пользователь обращается к приложению, которое хранит данные в базе данных. Однако для доступа к базе данных используется проверка собственно приложения, но не пользователя. Как правило, приложение проверяет подлинность отдельных пользователей на основе учетных данных, которые они предоставляют при входе в систему, или на основании учетных данных Windows. В этом случае сервер базы данных (в этом примере – экземпляр SQL Server), не должен содержать имена входа для отдельных пользователей приложения, веб-узла или службы. Необходимо иметь имя входа только для приложения, так как ему доверяется аутентификация и авторизация своих собственных пользователей перед доступом к данным.

Модель безопасности олицетворения или делегирования полномочий, представленная на рисунке 10, обычно используется в бизнес-приложениях.

Этот альтернативный подход для доступа к серверу базы данных использует учетные данные пользователя. Приложение может выполняться в контексте собственной учетной записи службы, но при подключении к серверу базы данных он олицетворяет пользователя, для которого он обращается к

данным. Таким образом, эта модель требует, чтобы экземпляр SQL Server содержал имя входа для каждого отдельного пользователя приложения с потенциально различными требованиями разрешений.

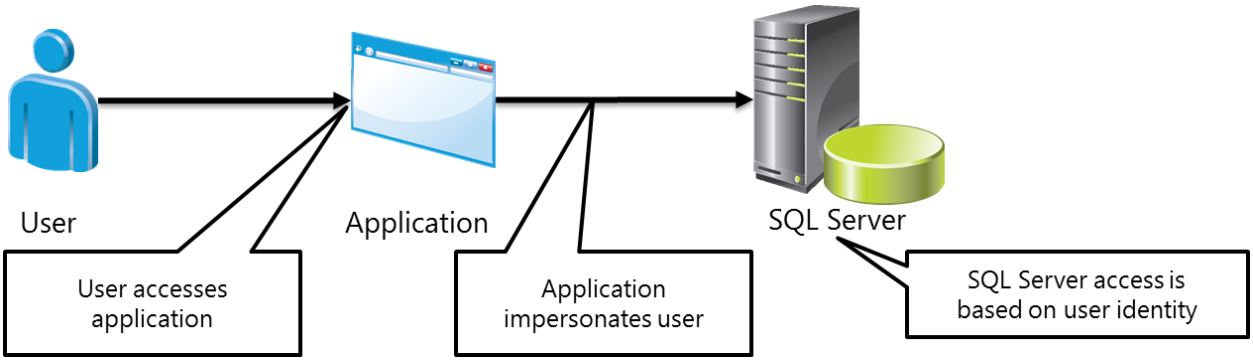

Рисунок 10 Модель безопасности олицетворения/делегирования полномочий

Обычно клиенты вызывают службы, чтобы платформа службы выполнила какие-либо действия от имени клиента. Олицетворение позволяет службе выступать в качестве клиента при выполнении такого действия. Делегирование позволяет внешней службе перенаправить запрос клиента внутренней службе таким образом, чтобы внутренняя служба также могла олицетворять клиента. Олицетворение чаще всего используется, чтобы проверить: имеет ли клиент разрешение на выполнение определенного действия. А делегирование позволяет передать возможности олицетворения (в том числе удостоверение клиента) внутренней службе.

В техническом плане приложение, которое выполняет действия, основанные на учетных данных пользователя, использует олицетворения, чтобы получить доступ к экземпляру SQL Server на том же сервере, а для доступа к SQL Server на удаленном сервере использует делегирование.

Параметры проверки подлинности SQL Server

Выбор модели безопасности приложений поможет запланировать количество отдельных имен входа, которые надо создать и обслуживать. Затем необходимо определить, каким образом эти имена входа будут проходить проверку подлинности. Как уже упоминалось, SQL Server поддерживает имена входа Windows и имена входа SQL Server.

SQL Server поддерживает два режима проверки подлинности:

- **Режим проверки подлинности Windows** является режимом по умолчанию. Поскольку эта модель безопасности SQL Server тесно интегрирована с Windows, часто ее называют встроенной функцией безопасности. Только пользователи с логинами Windows могут подключаться к SQL Server. Пользователи Windows, прошедшие проверку подлинности, не должны предъявлять дополнительные учетные данные.
- **Режим смешанной аутентификации** поддерживает проверку подлинности как средствами Windows, так и средствами SQL Server. Пары имен пользователей и паролей ведутся в SQL Server.

Какой тип проверки подлинности будет использоваться, можно указать при установке SQL Server. Можно изменить режим после установки, хотя это потребует перезапуска сервера, чтобы изменения вступили в силу.

Если при установке SQL Server использовали смешанный режим проверки подлинности, программа установки позволяет назвать имя входа SQL Server «sa». Важно создать сложный пароль для этого имени входа, поскольку он имеет права администратора на уровне сервера баз данных.
Если при установке SQL Server использовали режим проверки подлинности Windows, то изменение его на смешанный режим аутентификации не позволит имя «sa» использовать как логин SQL Server.

Управление именами входа

Имя входа – это субъект безопасности, с помощью которого система безопасности может проверить подлинность лица или сущности. Имя входа необходимо пользователю для соединения с SQL Server. Вы можете создать имя входа на основе участника Windows (например, пользователя или группы домена Windows), либо на основе пользователя, не являющегося участником Windows (например, имени входа SQL Server).

Можно создавать имена входа в среде SSMS или с помощью кода T-SQL. Использование сценария Transact-SQL намного быстрее и удобнее.

Создание имен входа с помощью инструкции CREATE LOGIN.

В этом примере создается имя входа ADVENTUREWORKS\SalesReps для группы Windows с таким же именем. База данных по умолчанию для пользователя будет salesdb. Если этот параметр не задан, база данных по умолчанию устанавливается master.

```
CREATE LOGIN [ADVENTUREWORKS\SalesReps]
FROM WINDOWS
WITH DEFAULT_DATABASE = [salesdb];
```

Имена пользователей и групп Windows должны заключаться в квадратные скобки, поскольку они содержат символ обратной косой черты.

Имена входа SQL Server создаются таким же образом. Однако, есть дополнительные аргументы, которые относятся только к именам входа SQL Server (например, аргумент PASSWORD).

В следующем примере показано, как создать логин с именем DanDrayton и паролем Pa$$w0rd:

```
CREATE LOGIN DanDrayton
WITH PASSWORD = 'Pa$$w0rd',
DEFAULT_DATABASE = [salesdb];
```

Политика безопасности имени входа SQL Server

Администраторы Windows могут включить политики для пользователей Windows, которые обеспечивают требования сложности паролей и истечения срока действия. SQL Server может применять аналогичные ограничения.

При создании логина SQL Server можно указать следующие параметры:

- MUST_CHANGE: при следующем входе в систему SQL Server предложит пользователю изменить пароль. Вы должны убедиться, что это сработает независимо от клиентского приложения, которое пользователь использует для подключения к SQL Server. Значение по умолчанию для этого параметра при использовании среды SSMS «ON», но при выполнении инструкции CREATE LOGIN значение «OFF».
- CHECK_POLICY = {ON | OFF}: Значение по умолчанию для этого параметра имеет значение ON, что усиливает политику сложности паролей для этого пользователя.
- CHECK_EXPIRATION = {ON | OFF}: Установка этого значения в «ON» заставляет пользователя изменять свой пароль через регулярные промежутки времени Значение по умолчанию для этого параметра при использовании среды SSMS «ON», но при выполнении инструкции CREATE LOGIN значение «OFF».

Можно настроить параметры политики для имени входа SQL Server в среде SSMS, в инструкции CREATE LOGIN или ALTER LOGIN.

Полное применение политики учетных записей не всегда является желательным. Например, некоторые приложения используют фиксированные учетные данные для подключения к серверу. Часто эти приложения не поддерживают регулярное изменение паролей входа. В этих случаях надо отключить истечение срока действия пароля для этих имен входа.

В следующем примере кода изменяем имя входа DanDrayton, созданное ранее, чтобы явно отключить проверку политики:

```
ALTER LOGIN DanDrayton
WITH CHECK_POLICY = OFF, CHECK_EXPIRATION = OFF;
```

Пароли можно изменять, например, с помощью инструкции ALTER LOGIN:

```
ALTER LOGIN DanDrayton
WITH OLD_PASSWORD = 'Pa$$w0rd',
PASSWORD = 'NewPa$$w0rd';
```

Если имена входа не используются, их можно удалить с сервера.

```
DROP LOGIN DanDrayton;
```

Если имена входа не используются только определенный период времени, следует их не удалять, а отключить и затем повторно включить:

```
ALTER LOGIN DanDrayton DISABLE;
...
ALTER LOGIN DanDrayton ENABLE;
```

Нельзя удалить текущее имя входа. Также нельзя удалить имя входа, владеющее любым защищаемым объектом уровня сервера или заданием агента SQL Server. Можно удалить имена входа, сопоставленные пользователям базы данных; однако это приведет к появлению пользователей, утративших связь с учетными записями.

6.3 Управление участниками уровня Базы данных

После создания имен входа, необходимо предусмотреть, чтобы хотя бы один логин имел доступ к базам данных. Доступ к базе данных предоставляется для логина путем создания для него пользователя базы данных.

Пользователи Базы данных

При создании пользователя базы данных ему назначается идентификатор (principal_id) и идентификатор безопасности (SID).

Можно создать пользователей базы данных с помощью инструкций CREATE USER или среды SSMS.

В следующем примере в БД *salesdb* создаются 3 пользователя:

```
USE salesdb;
CREATE USER SalesReps
     FOR LOGIN [ADVENTUREWORKS\SalesReps]
     WITH DEFAULT_SCHEMA = Sales;
CREATE USER DanDrayton
     FOR LOGIN DanDrayton;
CREATE USER WebUser
     FOR LOGIN [ADVENTUREWORKS\WebAppSvcAcct];
```

Обратите внимание, что первый оператор CREATE USER включает в себя схему по умолчанию. Схемы – это пространства имен, которые используются для упорядочивания объектов в базе данных. Если параметр WITH DEFAULT_SCHEMA не задан, то схемой по умолчанию будет встроенная схема dbo. В третьем операторе имя пользователя отличается от имени входа, с которым он связан.

Можно удалить пользователей из базы данных (с помощью инструкции DROP USER или среды SSMS). Однако нельзя удалить пользователя базы данных, который является владельцем любого защищаемого объекта (например, таблицы или представления).

Управление идентификаторами безопасности (SID)

При создании имени входа SQL Server, ему назначаются идентификатор (principal_id) и идентификатор безопасности (SID). Когда затем создается пользователь базы данных, principal_id и SID логина вносятся в системное представление *sys.sysusers*. Если затем выполнить резервное копирование и восстановление базы данных на другой сервер, пользователь по-прежнему присутствует в базе данных, но без соответствующего ему имени входа на сервере. Даже если затем создать новый логин с тем же именем (как у пользователя в базе данных), это не будет работать потому, что имя входа будет иметь другой SID.

Устранить проблему можно, если с помощью инструкции ALTER USER обновить пользователя базы данных, чтобы связать его с новым логином:

```
ALTER USER DanDrayton WITH LOGIN = DanDrayton;
```

Это решает проблему, но, если впоследствии восстановить базу данных на том же или другом сервере, то проблема возникнет снова. Лучший способ избежать этой проблемы при создании имени входа использовать предложение SID.

Управление доступом dbo и guest

Имя входа не может получить доступ к базе данных, если явным образом ему не предоставлен доступ через создание пользователя базы данных. Это утверждение в целом верно, но есть два исключения: каждая база данных SQL Server включает пользователей dbo и guest.

Пользователь dbo – это специальный пользователь (или владелец базы данных) – представляет собой учетную запись пользователя, которая имеет неявно заданные разрешения на выполнение любых действий с базой данных. Он всегда присутствует в каждой базе данных, его невозможно удалить. Любой член фиксированной серверной роли sysadmin (включая пользователя sa при использовании смешанного режима проверки подлинности), который использует базу данных, сопоставляется с этим пользователем.

Владелец базы данных может быть изменен. Любой пользователь (имя входа SQL Server или Microsoft Windows), имеющий возможность соединения с SQL Server, может стать владельцем базы данных. В следующем примере показано, как изменить владельца базы данных с помощью инструкции ALTER AUTHORIZATION:

```
ALTER AUTHORIZATION ON DATABASE::salesdb
     TO [ADVENTUREWORKS\Database_Managers];
```

---

*Учетную запись пользователя **dbo** часто путают с предопределенной ролью базы данных **db_owner**. Членство в роли **db_owner** не дает доступа к пользовательским привилегиям роли **dbo**.*

Учетная запись пользователя guest является встроенной учетной записью во всех версиях SQL Server. После прохождения пользователем проверки подлинности и получения разрешения на вход в экземпляр SQL Server в каждой базе данных должна существовать отдельная учетная запись пользователя, к которой пользователь должен получить доступ. Это требование предотвращает соединение пользователей с экземпляром SQL Server и получение доступа ко всем базам данных сервера. Наличие учетной записи пользователя guest в базе данных дает возможность обойти данное требование Она позволяет пользователю с именем входа, которое не сопоставлено ни одному пользователю в конкретной базе данных, получить доступ к этой базе данных.

Нельзя удалить учетную запись пользователя guest, но можно ее отключить, чтобы предотвратить доступ к БД:

```
REVOKE CONNECT FROM guest;
```

По умолчанию учетная запись guest в новых БД отключена. Можно включить гостевую учетную запись с помощью инструкции GRANT CONNECT:

```
GRANT CONNECT TO guest;
```

По умолчанию учетная запись пользователя guest включена в системных БД master, msdb и tempdb. Вы не должны пытаться отменить гостевой доступ в этих базах данных.

Управление ролями базы данных

Роли базы данных позволяют сгруппировать пользователей по требованиям доступа к ресурсам БД или правам.

Существует два типа ролей уровня базы данных: предопределенные роли базы данных, являющиеся стандартными для баз данных, и пользовательские роли базы данных, которые можно создавать.

Каждая база данных включает встроенные фиксированные роли базы данных с заранее определёнными правами для наиболее распространенных сценариев. Описание этих ролей приведено в таблице 11.

Таблица 11. Фиксированные роли уровня базы данных

Имя роли	Описание
db_owner	Члены предопределенной роли **db_owner** могут выполнять все действия по настройке и обслуживанию базы данных, а также удалять базу данных.
db_securityadmin	Члены предопределенной роли **db_securityadmin** могут изменять членство в роли и управлять разрешениями. Добавление участников к этой роли может привести к непреднамеренному повышению прав доступа.
db_accessadmin	Члены предопределенной роли **db_accessadmin** могут добавлять или удалять права удаленного доступа к базе данных для имен входа и групп Windows, а также имен входа SQL Server .
db_backupoperator	Члены предопределенной роли **db_backupoperator** могут создавать резервные копии базы данных.
db_ddladmin	Члены предопределенной роли **db_ddladmin** могут выполнять любые команды языка определения данных (DDL) в базы данных.
db_datawriter	Члены предопределенной роли **db_datawriter** могут добавлять, удалять или изменять данные во всех пользовательских таблицах.
db_datareader	Члены предопределенной роли **db_datareader** могут считывать все данные из всех пользовательских таблиц.
db_denydatawriter	Члены предопределенной роли **db_denydatawriter** не могут добавлять, изменять или удалять данные в пользовательских таблицах базы данных.
db_denydatareader	**db_denydatareader** не могут считывать данные из пользовательских таблиц базы данных.
Public	Члены этой роли могут выполнять просмотр любых объектов БД. Каждый пользователь базы данных является членом роли базы данных public. Если для пользователя не были предоставлены или запрещены конкретные разрешения на защищаемый объект, он наследует разрешения роли public на этот объект. Пользователей базы данных нельзя удалить из роли public.

Можно также создать пользовательские роли базы данных, которые позволят установить более детальные разрешения. Это можно сделать с помощью SSMS или инструкцию CREATE ROLE, как показано в следующем примере:

```
CREATE ROLE product_reader;
```

С помощью среды SSMS или инструкции ALTER ROLE можно управлять членством в роли, например, добавлять и удалять имена входа. В следующем примере кода пользователь веб-приложение добавляется к product_reader роль:

```
ALTER ROLE product_reader
     ADD MEMBER WebApp;
```

Чтобы удалить члена роли, используйте оператор ALTER ROLE с предложением DROP MEMBER.

Запрос к системному представлению sys.database_role_members возвращает список всех членов фиксированных и пользовательских ролей базы данных.

Не добавляйте гибкие роли базы данных в качестве членов предопределенных ролей. Это может привести к непреднамеренному повышению прав доступа.

Управление ролями приложений

Роль приложения – это участник безопасности базы данных, который позволяет приложению активировать альтернативный (обычно повышенный) контекст безопасности для получения разрешений, необходимых для конкретной операции, которые не предоставляются текущему пользователю.

Например, приложение «пункт продажи» используется в супермаркете для записи сделок купли-продажи. Оператор, оформляющий заказ, может войти, используя свои собственные учетные данные, и ввести данные каждого закупаемого товара. Для этого ему требуется разрешение INSERT для таблицы сделок купли-продажи. В некоторых случаях после завершения сделки клиенту может понадобиться добавить дополнительный элемент. Это требует разрешение UPDATE в таблице сделок купли-продажи, которое не было предоставлено оператору кассы. В этом случае контролер может ввести код, чтобы подтвердить операцию, после чего приложение активизирует роль приложения, которое имеет необходимые разрешения. После того, как обновление было выполнено, роль приложения можно отключить, и контекст безопасности возвращается к учетной записи оператора на кассе.

Создать роль приложения можно с помощью инструкции Transact-SQL CREATE APPLICATION ROLE, указав пароль:

```
CREATE APPLICATION ROLE sales_supervisor
     WITH PASSWORD = 'Pa$$w0rd';
```

После того, как роль приложения создана, и ей назначены необходимые разрешения, она может быть активирована путем выполнения системной хранимой процедуры sp_setapprole.

```
EXEC sp_setapprole 'SalesSupervisor', 'Pa$$w0rd';
GO
```

Роль приложения остается активной до тех пор, пока пользователь не разрывает соединение с SQL Server или не деактивируется с помощью системной хранимой процедуры sp_unsetapprole. Однако чтобы использовать sp_unsetapprole, необходимо указать файл cookie, который был создан при активации роли приложения.

В следующем примере кода показано, как создать cookie при активации роли приложения:

```
EXEC sp_setapprole 'sales_supervisor', 'Pa$$w0rd',
          @fCreateCookie = true, @cookie = @cookie OUTPUT;
```

Параметр **OUTPUT** cookie-файла для sp_setapprole в настоящее время описан в документации как **varbinary(8000)**, что верно определяет его максимальную длину.

Когда приложение завершит операцию, для которой требуются разрешения роли приложения, можно отключить роль приложений и вернуться к контексту безопасности текущего пользователя, выполнив процедуру sp_unsetapprole, указав файл cookie, который был создан при активации роли приложения:

```
EXEC sp_unsetapprole @cookie;
```

6.4 Управление разрешениями уровня Базы данных

После создания пользователей БД, что обеспечило доступ к базе данных, а также ролей БД, можно применить разрешения для управления доступом пользователей к данным и выполнять задачи в БД.

Большинство баз данных имеют более детализированные требования к безопасности, чем фиксированные роли базы данных. Надо стремиться использовать роли базы данных для групп пользователей и свести к минимуму число отдельных явных разрешений, которые необходимо назначать для защиты базы данных.

Разрешения уровня базы данных

База данных – это защищаемый объект, хранящийся на сервере, который является родителем базы данных в иерархии разрешений.

На уровне базы данных можно настроить разрешения на следующие защищаемые объекты:

- Пользователи (Users)
- Роли базы данных (Database roles)
- Роли приложений (Application roles)
- Полнотекстовые каталоги (Full text catalogs)
- Сертификаты (Certificates)
- Асимметричные ключи (Asymmetric keys)
- Симметричные ключи (Symmetric keys)
- Схемы (Schemas)

Кроме того можно настроить разрешения на некоторые объекты базы данных, которые содержатся в схемах:

- Таблицы (Tables)
- Функции (Functions)
- Хранимые процедуры (Stored procedures)
- Представления (Views)
- Индексы (Indexes)
- Ограничения (Constraints)

Разрешения можно использовать для выполнения операций DDL на конкретные защищаемые объекты. Например, в следующем примере роли базы данных sales_admin предоставлено разрешение на изменение роли приложения sales_supervisor:

```
GRANT ALTER ON APPLICATION ROLE::sales_supervisor
          TO sales_admin;
```

Можно также использовать разрешения для выполнения операций DML на объектах базы данных. В следующем примере предоставляется разрешение SELECT на таблицу dbo. ProductCategory и dbo.Product для роли базы данных product_reader:

```
GRANT SELECT ON OBJECT::dbo.ProductCategory
          TO product_reader;
GRANT SELECT ON dbo.Product TO product_reader;
```

Обратите внимание, что для объектов в схеме базы данных может быть использован префикс OBJECT::, но он не является обязательным.

Вы можете просмотреть действующие разрешения для пользователя или роли на вкладке Защищаемые объекты (Securables) диалогового окна Свойства (Properties) для данного пользователя или роли.

Управление Схемами

Схема (Schema) – это контейнер объектов, используемый для определения пространства имен объектов внутри базы данных. Схемы упрощают управление разрешениями за счет создания подмножества совместно управляемых объектов. Пользователя можно сделать владельцем схемы и назначить ему схему по умолчанию. Если пользователю назначена схема по умолчанию, при разрешении имен объектов их поиск в первую очередь выполняется в схеме по умолчанию. В противном случае используется схема по умолчанию для группы, к которой принадлежит пользователь. При определении схемы по умолчанию учитываются только те группы Windows, для которых были созданы связанные пользователи БД.

В базе данных есть несколько встроенных схем. Пользователи dbo и guest связаны со схемами, имеющими имена dbo и guest соответственно. Схема dbo используется, если SQL Server не может определить схему по умолчанию, которую нужно использовать, а также служит схемой по умолчанию для всех

членов роли sysadmin. Схемы sys и INFORMATION_SCHEMA зарезервированы для системных объектов, в этих схемах нельзя создавать и удалять объекты.

Можно создать свою схему с помощью инструкции CREATE SCHEMA:

```
CREATE SCHEMA sales;
```

Запрос к системному представлению sys.schemas возвращает список всех схем в базе данных.

Схемы и разрешение имен объектов. Имя схемы является составной частью идентификатора объектов. Полное имя любого объекта – это «*Server.Database.Schema.Object*».

При использовании кода Transact-SQL в контексте базы данных, имена объектов часто сокращают. Рекомендуется явно указывать схему и имя объекта, т.к. разные схемы могут содержать объекты с одинаковыми именами (например, *dbo.product, sales.product, production.product*).

Схемы и наследование разрешений. Вы можете контролировать доступ к объектам в схеме, установив явные разрешения на сами объекты, но схемы позволяют определять права доступа более простым способом, потому что любые права, которые применяются к схеме, неявно применяются и к объектам, которые она содержит. Например, если предоставить разрешение SELECT на схему, то неявно предоставляется разрешение SELECT на все объекты, которые поддерживают разрешение SELECT в схеме (таблицы и представления).

Следующий пример кода предоставляет разрешение INSERT для схемы sales роли базы данных sales_writer. Членам этой роли неявно будет предоставлено разрешение INSERT для всех таблиц и представлений в схеме sales:

```
GRANT INSERT ON SCHEMA::sales  TO sales_writer;
```

Разрешения для Таблиц и Представлений

К таблицам и представлениям применяются следующие разрешения DML:

- SELECT. Участникам необходимо это разрешение для извлечения данных из таблицы или представления с помощью инструкции SELECT.
- INSERT. Участникам необходимо это разрешение на добавление новых строк в таблицы или представления с помощью инструкции INSERT.
- UPDATE. Участникам необходимо это разрешение на изменение данных в таблице или представлении с помощью инструкции UPDATE.
- DELETE. Участникам необходимо это разрешение на удаление строк из таблицы или представления с помощью инструкции DELETE.
- REFERENCES. Участникам необходимо это разрешение для проверки ограничений целостности, например, при наличии ограничения внешнего ключа (*foreign-key*). Пользователь имеет разрешение на INSERT, UPDATE в таблице с внешним ключом, но на родительскую таблицу ему

не предоставляется разрешение (SELECT), чтобы ограничить доступ и не давать избыточных прав доступа. В этом случае на таблицу с первичным ключом ему нужно дать разрешение REFERENCES: он сможет производить действия с данными, проверка ограничений целостности выполнится. Это разрешение также можно предоставлять на отдельные столбцы таблицы и функции.

Помимо этих разрешений (DML), таблицы и представления имеют разрешения DDL и административные разрешения (ALTER, CONTROL, TAKE OWNERSHIP и VIEW DEFINITION).

Вы можете просмотреть в среде SSMS действующие разрешения выбранных пользователей и ролей на конкретный объект на странице Разрешения (Permissions) в диалоговом окне Свойства для этого объекта. Также можно просмотреть действующие разрешения на выбранные объекты для участника конкретной базы данных, просмотрев вкладку Защищаемые объекты (Securables) диалогового окна Свойства для этого участника (см. рисунок 11).

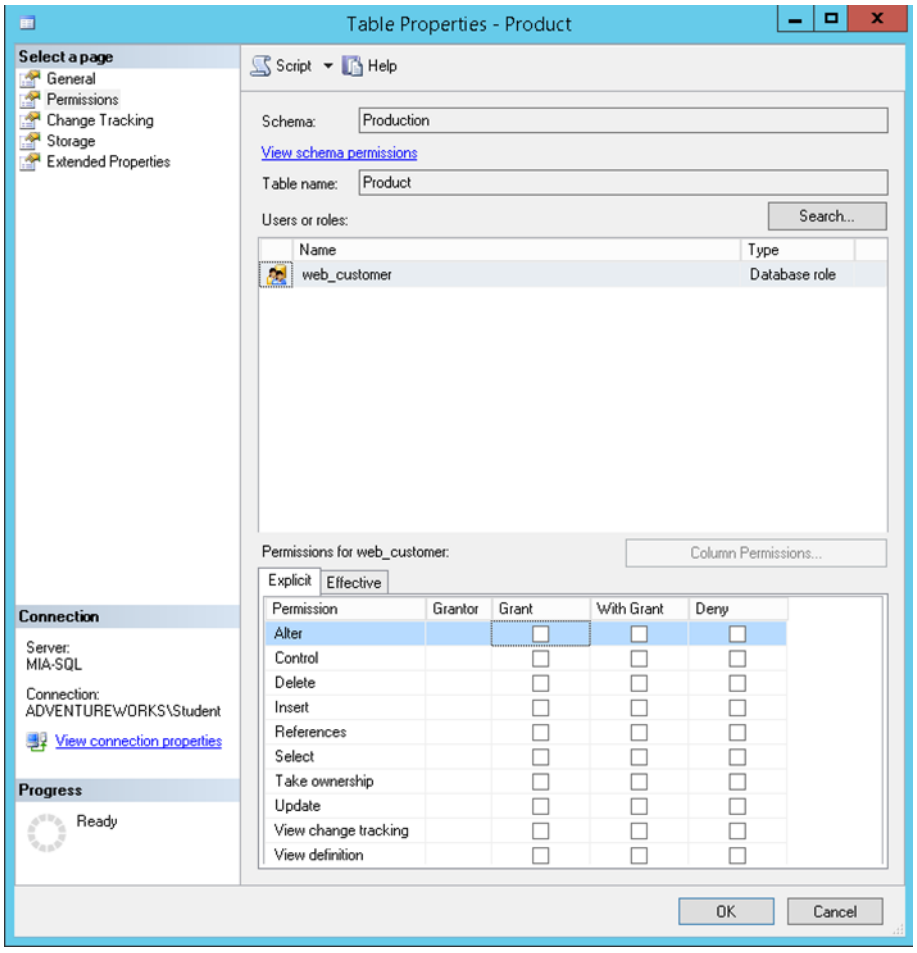

Рисунок 11 Просмотр действующих разрешений в окне Свойства объекта

Разрешения уровня столбца. В дополнение к назначению разрешений на таблицы или представления, можно также давать разрешения на уровне столбцов. Это обеспечивает более детальный уровень защиты данных в базе данных. При этом не нужно выполнять отдельные инструкции GRANT или

DENY для каждого столбца, где требуется назначить разрешения: можно предоставить список столбцов в одном операторе GRANT или DENY.

```
GRANT SELECT ON production.product (Name, Price)
     TO web_customer;
GO
DENY SELECT ON production.product (Cost) TO web_customer;
GO
```

Инструкция GRANT для разрешения на уровне столбцов имеет приоритет над инструкцией DENY на уровне объектов. Если для пользователя выполнить инструкцию DENY на уровне таблицы, а затем выполнить инструкцию GRANT на уровне столбцов, разрешение DENY удаляется, и пользователь может получить доступ к столбцам, к которым ему предоставлен доступ.

Однако если затем снова выполнить инструкцию DENY уровня таблицы, пользователю будет отказано в разрешении на таблицу, включая и те столбцы, к которым он ранее имел доступ.

Разрешения для исполняемого кода

В дополнение к предоставлению управления над тем, кто получает доступ к данным в базе данных или к объектам на сервере, SQL Server позволяет контролировать, какие пользователи могут выполнять код. Надлежащее управление безопасностью исполнения кода является важным аспектом вашей архитектуры безопасности.

Хранимые процедуры. По умолчанию пользователи не могут выполнять хранимые процедуры, созданные другими пользователями, если им не предоставлено разрешение EXECUTE на хранимую процедуру. Кроме того, могут потребоваться разрешения на доступ к объектам, которые хранимая процедура использует.

В следующем примере роль web_customer предоставляется разрешение на выполнение хранимые процедуры sales.insert_order:

```
GRANT EXECUTE ON sales.insert_order TO web_customer;
```

Определяемые пользователем функции. Необходимо также назначить пользователям разрешения для выполнения пользовательских функций (UDF). Разрешения, которые необходимо назначить, зависят от типа UDF, с которой вы работаете:

- Определяемые пользователем скалярные функции возвращают одно значение. Пользователям, имеющим доступ к этим функциям, требуется разрешение EXECUTE.
- Пользовательские функции, возвращающие табличные значения (TVF), возвращают не одно значение, а таблицу результатов. Доступ к TVF аналогичен разрешению на таблицу: требуется разрешение SELECT, но не EXECUTE.

- Редко, но возможно назначить разрешения INSERT, UPDATE и DELETE на TVF, известной как «inline TVF», которая может в некоторых случаях обновляться.

Кроме этих разрешений существуют сценарии, где необходимо также назначить разрешение REFERENCES для пользователей, чтобы они могли правильно выполнить UDF. Эти сценарии включают в себя функции, которые:

- Используются в ограничениях CHECK.
- Вычисляют значения ограничений по умолчанию.
- Используются в вычисляемых столбцах.

В следующем примере показано, как предоставить разрешения роли web_customer на выполнение функции dbo.calculate_tax:

```
GRANT EXECUTE ON dbo.calculate_tax TO web_customer;
```

Управляемый код – это код .NET Framework, который поставляется в сборках. Сборки могут существовать в виде DLL или EXE-файлов. Сборки можно только загружать с помощью интеграции со средой CLR SQL Server.

Сборки регистрируются в базе данных SQL Server с помощью инструкции CREATE ASSEMBLY. На основе сборки можно создавать (стандартные) объекты БД, для которых применяются (стандартные) разрешения объекта. Например, пользователям требуется разрешение EXECUTE для запуска хранимой процедуры независимо от того, создана она кодом Transact-SQL или из сборки.

Наборы разрешений. Независимо от того, какой код .NET Framework включен в сборку, действия, которые может выполнить код, определяются набором разрешений, заданным при создании сборки, параметром WITH PERMISSION_SET:

- WITH PERMISSION_SET = SAFE (по умолчанию) является наиболее ограниченным набором разрешений и блокирует доступ к внешним системным ресурсам. Код, исполняемый с разрешениями SAFE, не может получить доступ к внешним системным ресурсам, таким как файлы, сеть, переменные окружения или реестр. Код может получить доступ к локальному экземпляру SQL Server, используя прямой путь доступа, которое называется контекстное соединение.
- WITH PERMISSION_SET = EXTERNAL_ACCESS позволяет сборкам получать доступ к внешним системным ресурсам, таким как файлы, сети, переменные окружения и реестр. Это разрешение даже необходимо для доступа к одному экземпляру SQL Server, если подключение осуществляется через сетевой интерфейс.
- WITH PERMISSION_SET = UNSAFE предоставляет сборкам неограниченный доступ к ресурсам как внутри, так и вне экземпляра SQL Server. Код, исполняемый из сборки с набором прав UNSAFE,

может вызывать неуправляемый код. Обратите внимание, что это разрешение в SSMS отображается как неограниченное (Unrestricted).

Наборы разрешений EXTERNAL_ACCESS и UNSAFE недоступны в автономной базе данных. Они требуют дополнительной настройки. Нельзя просто указать параметр EXTERNAL_ACCESS при выполнении инструкции CREATE ASSEMBLY. До этого необходимо пометить базу данных, как TRUSTWORTHY (что не рекомендуется!) или создать асимметричный ключ из файла сборки в базе данных master, создайть имя входа, которое сопоставляется с ключом и предоставить имени входа разрешение EXTERNAL ACCESS ASSEMBLY на сборку.

Цепочки владения

Все объекты базы данных имеют владельцев. Рекомендуется, чтобы все объекты в схеме принадлежали владельцу схемы, т.к. наличие одного владельца для всех объектов в схеме (которая сама имеет владельца) упрощает управление разрешениями. В таком случае объект со свойством principal_id, равным NULL, наследует свое право собственности от схемы, где он находится.

Если несколько объектов базы данных последовательно обращаются друг к другу, устанавливается цепочка владения. Цепочки владения применяются для представлений, хранимых процедур и функций.

Если все объекты в цепочке имеют одного владельца, то образуется **неразрывная цепочка владения**. В неразрывных цепочках владения доступ к нижележащим объектам разрешен, когда разрешен доступ к объектам верхнего уровня. Это позволяет управлять доступом к нескольким объектам, назначая разрешения только одному объекту, и обеспечивает небольшое повышение производительности (в случаях, когда позволено пропускать проверку наличия разрешений).

Как выполняется проверка разрешений в цепочке. Когда доступ к объекту производится через цепочку, SQL Server сначала сравнивает владельца объекта с владельцем вызывающего объекта (предшествующее звено в цепи). Если оба объекта имеют одного владельца, то разрешения для объекта не проверяются. Рассмотрим пример, представленный на рисунке 12.

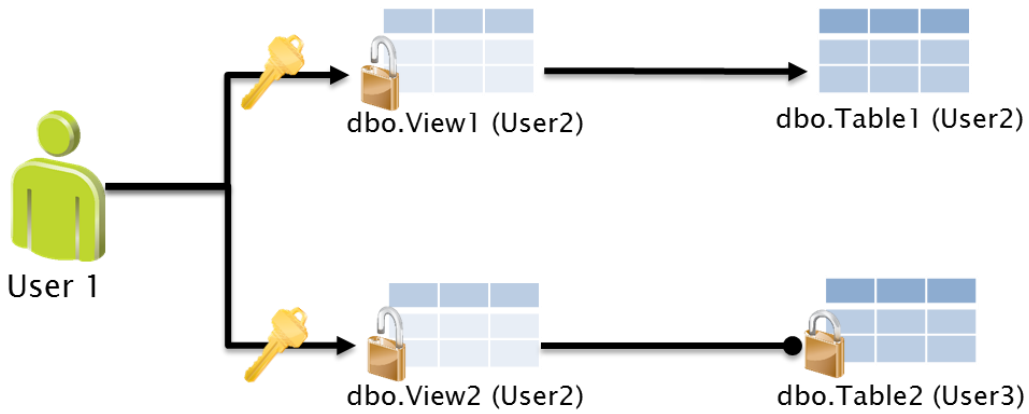

Рисунок 12 Проверка разрешений в цепочке владений

1. Рассмотрим первую (верхнюю) цепочку владения:

   User1 не имеет разрешений на таблицу *dbo.Table1* (владелец User2). Пользователь User2 создает представление *dbo.View1*, которое обращается к таблице *dbo.Table1* (владелец User2), и дает разрешение пользователю User1 на доступ к *dbo.View1*. Так как User2 является владельцем объекта верхнего уровня (представление) и базового объекта (таблица), то пользователю User1 предоставляется доступ к *dbo.View1*.
2. Рассмотрим вторую (нижнюю) цепочку владения:

   Затем пользователь User2 создает представление *dbo.View2*, которое обращается к таблице *dbo.Table2* (владелец User3).

   Даже если User1 имеет разрешение пользователя User2 на представление *dbo.View2*, ему будет отказано в доступе из-за сломанной цепочки владения от объекта верхнего уровня (представление) к базовому объекту (таблица).

   Тем не менее, если будут получены полные разрешения на это представление, а именно: пользователь User2 даст разрешения на представление *dbo.View2*, а пользователь User3 даст разрешения на базовую таблицу dbo.Table2, то пользователь User1 сможет получить доступ к представлению *dbo.View2*.

6.5 Шифрование Баз данный

Под безопасностью обычно понимают средства защиты от хакеров и другие риски подключения к данным. При этом часто упускается из виду риск физической кражи носителей данных. Поэтому соблюдение политики безопасности обязывает использовать шифрование, как способ скрытия данных с помощью ключа или пароля. Это делает данные бесполезными без соответствующего ключа или пароля для дешифрования. Шифрование не решает проблемы управления доступом, однако повышает защиту за счет ограничения потери данных даже при обходе системы управления доступом.

SQL Server 2014 включает в себя два способа шифрования данных: прозрачное шифрование данных (Transparent Data Encryption, TDE) и расширенное управление ключами (Extensible Key Management EKM).

SQL Server шифрует данные, используя иерархическую структуру средств шифрования и управления ключами. На каждом уровне данные низшего уровня шифруются на основе комбинации сертификатов, асимметричных ключей и симметричных ключей. Асимметричные и симметричные ключи можно хранить вне модуля расширенного управления ключами SQL Server.

В SQL Server можно шифровать соединения, данные и хранимые процедуры.

Несмотря на то, что шифрование является полезным средством обеспечения безопасности, его не следует применять ко всем данным или соединениям: шифрование может потребоваться, если пользователи получают доступ к данным через открытую сеть. Использование шифрования включает политику управления паролями, ключами и сертификатами.

Обзор прозрачного шифрования данных

Прозрачное шифрование данных предоставляет дополнительный уровень защиты инфраструктуры базы данных путем шифрования файлов данных и файлов журнала без необходимости перенастройки клиентских приложений или дополнительных усилий от разработчика. Функция прозрачного шифрования данных защищает "неактивные" данные, то есть файлы данных и журналов, но не обеспечивает шифрование каналов связи.

Сам экземпляр SQL Server выполняет в реальном времени шифрование и дешифрование данных, поэтому этот процесс является прозрачным для приложений и пользователей. Когда SQL Server выполняет запись страниц данных на диск, они шифруются; а когда страницы считываются в память, они дешифруются. Прозрачное шифрование данных не увеличивает размер зашифрованной базы данных.

На рисунке 13 показана архитектура прозрачного шифрования данных.

Главный ключ службы
(Service Master Key, SMK)

- Создается во время установки SQL Server
- Шифрование данных по Windows DPAPI

Мастер Ключ базы данных
(Database Master Key, DMK)

- Создается в системной базе данных **master**
- Шифрование данных по SMK

Сертификат сервера
(Server Certificate)

- Создается в системной базе данных **master**
- Шифрование данных по *DMK*

Ключ шифрования базы данных
(Database Encryption Key, DEK)

- Создается в пользовательской базе данных (User DB)
- Шифрование данных с помощью сертификата сервера (*server certificate*)

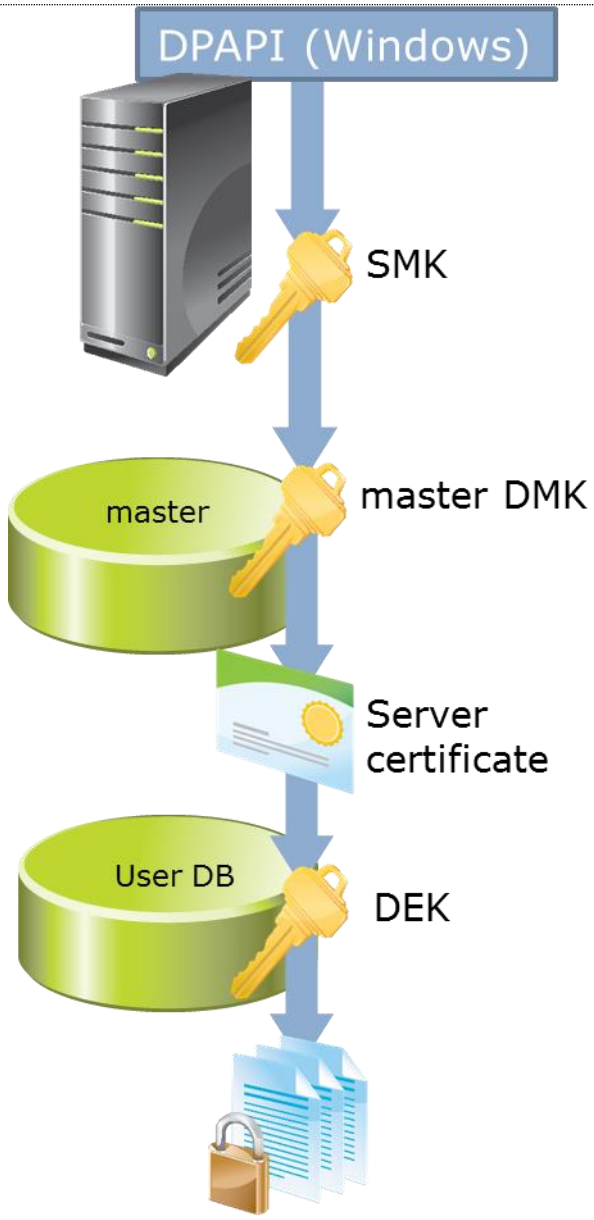

Рисунок 13 Архитектура прозрачного шифрования данных

При шифровании используется ключ шифрования базы данных (DEK), который хранится в загрузочной записи базы данных, где можно получить к нему доступ при восстановлении. Ключ шифрования базы данных является симметричным ключом, защищенным сертификатом, который хранится в базе данных **master**. Это позволяет разработчикам программного обеспечения шифровать данные с помощью алгоритмов шифрования AES и 3DES, не меняя существующие приложения.

Настройка прозрачного шифрования данных

Чтобы включить TDE, необходимо выполнить следующие действия:

1. Создание мастер-ключа базы данных в базе данных **master**.
2. Создание сертификата сервера в базе данных **master**.

```
USE master
GO
CREATE MASTER KEY ENCRYPTION
     BY PASSWORD = 'Pa$$w0rd';
GO
CREATE CERTIFICATE  Security_Certificate
     WITH SUBJECT = ' DEK_Certificate';
GO
```

Как только вы создали сертификат сервера и связанный с ним закрытый ключ, необходимо создать их резервную копию. Это минимизирует риск потери данных. В случае если сертификат окажется недоступен или понадобится восстановить базу данных на другом сервере либо присоединить ее к другому серверу, необходимо иметь копии сертификата и закрытого ключа. Иначе будет невозможно открыть базу данных. Сертификат следует сохранить, даже если функция прозрачного шифрования в БД будет отключена, т.к. части журнала транзакций могут по-прежнему оставаться защищенными, и для некоторых операций будет требоваться сертификат (до выполнения полного резервного копирования базы данных). Сертификат, который превысил свой срок хранения, все еще может быть использован при прозрачном шифровании данных.

В следующем примере кода показано, как создать резервную копию сертификата сервера и его закрытого ключа:

```
BACKUP CERTIFICATE Security_Certificate
     TO FILE = 'D:\backups\security_certificate.cer'
     WITH PRIVATE KEY
     (FILE = 'D:\backups\security_certificate.key' ,
     ENCRYPTION BY PASSWORD = 'CertPa$$w0rd');
```

3. Создание ключа шифрования базы данных в базе данных пользователя, которую требуется зашифровать:

```
USE AdventureWorks
GO
CREATE DATABASE ENCRYPTION KEY
     WITH ALGORITHM = AES_128
     ENCRYPTION BY SERVER CERTIFICATE Security_Certificate
GO
```

4. Включить шифрование для пользовательской базы данных:

```
ALTER DATABASE AdventureWorks2012
     SET ENCRYPTION ON;
```

Чтобы проверить, выполняется ли шифрование базы данных, можно выполнить запрос:

```
USE master;
SELECT name, is_encrypted FROM sys.databases ;
```

В столбце name выводится имя БД, а значение в столбце is_encrypted указывает: зашифрована (1) или не зашифрована (0) база данных.

Перемещение зашифрованных баз данных

Основной причиной шифрования базы данных является предотвращение несанкционированного доступа к данным. Нельзя просто присоединить файлы из зашифрованной базы данных на другой сервер и читать данные.

Если необходимо переместить зашифрованную базу данных на другой сервер, необходимо также переместить соответствующие ключи и сертификаты [5].

Ниже перечислены шаги для перемещения базы данных с поддержкой TDE:

1. На исходном сервере отключите базу данных, которую требуется переместить.
2. Скопируйте или переместите файлы базы данных в то же расположение на целевом сервере.
3. Создайте DMK в базе данных master на целевом сервере.
4. Используйте инструкцию CREATE CERTIFICATE для создания сертификата сервера на сервере назначения из резервной копии исходного (оригинального) сертификата сервера и его закрытого ключа.
5. Присоедините базу данных на целевом сервере.

Расширенное управление ключами

В корпоративной среде с соблюдением требований высокого уровня безопасности управление ключами шифрования на уровне отдельного сервера базы данных может оказаться непрактичным [5].

Многие организации приняли корпоративное решение, которое позволяет им управлять ключами шифрования и сертификатов с помощью поставщика аппаратных модулей безопасности (hardware security modules, HSMs) для безопасного хранения ключей. SQL Server поддерживает расширенное управление ключами (EKM), что дает возможность регистрировать модули от сторонних поставщиков в SQL Server и позволяет использовать ключи шифрования, хранящиеся на них.

Чтобы использовать ключи от стороннего поставщика EKM для реализации TDE, необходимо выполнить следующие задачи:

1. По умолчанию расширенное управление ключами отключено. Чтобы включить его, воспользуйтесь системной хранимой процедурой sp_configure с параметрами, приведенными в следующем примере:

```
sp_configure 'show advanced options', 1
GO
RECONFIGURE
GO
sp_configure 'EKM provider enabled', 1
GO
RECONFIGURE
GO
```

2. Создание провайдера служб шифрования из файла, предоставляемого поставщиком EKM:

```
CREATE CRYPTOGRAPHIC PROVIDER EKM_Provider
FROM FILE = 'S:\EKM_Files\EKMKey.dll' ;
```

3. Создание учетной записи для системных администраторов:

```
CREATE CREDENTIAL EKM_Credential
WITH IDENTITY = 'EKM_Admin',
SECRET = 'Pa$$w0rd'
FOR CRYPTOGRAPHIC PROVIDER EKM_Provider ;
```

4. Добавьте учетную запись для имени входа, которое будет использоваться для настройки шифрования:

```
ALTER LOGIN [ADVENTUREWORKS\DBAdmin]
ADD CREDENTIAL EKM_Credential ;
```

5. Создайте асимметричный ключ, хранящийся в поставщике EKM:

```
USE master ;
GO
CREATE ASYMMETRIC KEY EKM_Login_Key
FROM PROVIDER EKM_Provider
WITH ALGORITHM = RSA_512,
PROVIDER_KEY_NAME = 'SQL_Server_Key' ;
```

6. Создание учетных данных для компонента Database Engine для использования при выполнении шифрования и расшифровки.

```
CREATE CREDENTIAL EKM_TDE_Credential
WITH IDENTITY = 'TDE_DB',
SECRET = 'Pa$$w0rd'
FOR CRYPTOGRAPHIC PROVIDER EKM_Provider ;
```

7. Создайте имя входа, используемое при TDE и добавить учетные данные.

```
CREATE LOGIN EKM_Login
FROM ASYMMETRIC KEY EKM_Login_Key ;
GO
ALTER LOGIN EKM_Login
ADD CREDENTIAL EKM_TDE_Credential ;
```

8. Создание ключа шифрования базы данных для шифрования базы данных.

```
USE AdventureWorks ;
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM  = AES_128
ENCRYPTION BY SERVER ASYMMETRIC KEY EKM_Login_Key ;
```

9. Включите TDE для базы данных.

```
ALTER DATABASE AdventureWorks
SET ENCRYPTION ON ;
```

Литература

1. Электронная документация по SQL Server 2014 - MSDN - Microsoft [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/ms130214(v=sql.120).aspx, свободный. — Яз. русский. (дата обращения: 01.11.2016).
2. Библиотека Microsoft SQL Server - MSDN [Электронный ресурс]. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/bb545450.aspx, свободный. — Яз. русский. (дата обращения: 01.11.2016).
3. Microsoft TechNet [Электронный ресурс]. – Режим доступа: https://technet.microsoft.com/ru-ru/, свободный. — Яз. русский. (дата обращения: 01.11.2016).
4. Станек Уильям Р. Microsoft SQL Server 2012. Справочник администратора / Пер. с англ. — М.: Издательство «Русская редакция»; СПб: БХВ-Петербург, 2013. — 576 с.: ил.
5. #20462C Administering Microsoft® SQL Server®. Официальный курс Microsoft.

Миссия университета – генерация передовых знаний, внедрение инновационных разработок и подготовка элитных кадров, способных действовать в условиях быстро меняющегося мира и обеспечивать опережающее развитие науки, технологий и других областей для содействия решению актуальных задач.

КАФЕДРА ПРОГРАММНЫХ СИСТЕМ

http://ps.ifmo.ru

Кафедра Программных систем входит в состав факультета Инфокоммуникационных технологий. На кафедре обеспечена возможность обучения студентов по современным образовательным стандартам в области программного обеспечения ИКТ. Для этого на кафедре работает высококвалифицированный преподавательский состав, имеется современная техническая база и специализированные лаборатории, оснащенные необходимым оборудованием и программным обеспечением; качественная методическая поддержка образовательных программ. Наши студенты принимают активное участие в российских и зарубежных исследованиях и конференциях, имеют возможность публикации результатов своих исследований в ведущих российских и зарубежных реферируемых изданиях. Существенным преимуществом является возможность выпускников продолжить научную деятельность в аспирантуре Университета ИТМО и в других передовых российских и зарубежных научных Центрах.

Кафедра обеспечивает подготовку бакалавров и магистров по направлениям:

- 11.03.02 Интеллектуальные инфокоммуникационные системы - бакалавры;
- 11.04.02 Программное обеспечение в инфокоммуникациях – магистры.

На кафедре реализуется международная образовательная программа DD Master Program, в рамках которой выпускники имеют возможность получить два диплома: Диплом Университета ИТМО с присвоением магистерской степени по направлению «Программное обеспечение в инфокоммуникациях» и Международный диплом - Master of Science in Technology Lappeenranta University of Technology in the field of Computer Science majoring in Software Engineering.

Выпускники кафедры обладают компетенциями:

- проектирования и создания рациональных структур ИКС;
- разработки алгоритмов решения задач и их программной реализации на основе современных платформ;

- моделирования процессов функционирования сложных систем;
  •обеспечения безопасности работы ИКС;
- реализации сетевых услуг и сервисов в ИКС;
- проектирования и разработки баз данных;
- разработки клиент-серверных приложений ИКС;
- проектирования, создания и поддержки Web-приложений;
- управления проектами перспективных направлений развития ИКС.

Трудоустройство выпускников кафедры возможно на предприятиях: ООО «Digital Design»; ООО «Аркадия»; ОАО «Ростелеком»; ООО «ЭПАМ Системз»; ООО «Т-Системс СиАйЭс» и многие другие.

Мы готовим квалифицированных инженеров в области инфокоммуникационных технологий с новыми знаниями, образом мышления и способностями быстрой адаптации к современным условиям труда.

Осетрова Ирина Станиславовна

Разработка баз данных в MS SQL Server 2014

Учебное пособие

В авторской редакции
Редакционно-издательский отдел Университета ИТМО
Зав. РИО Н.Ф. Гусарова
Подписано к печати
Заказ №
Тираж 100 экз.
Отпечатано на ризографе

Редакционно-издательский отдел
Университета ИТМО
197101, Санкт-Петербург, Кронверкский пр., 49