**Администрирование баз данных**

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

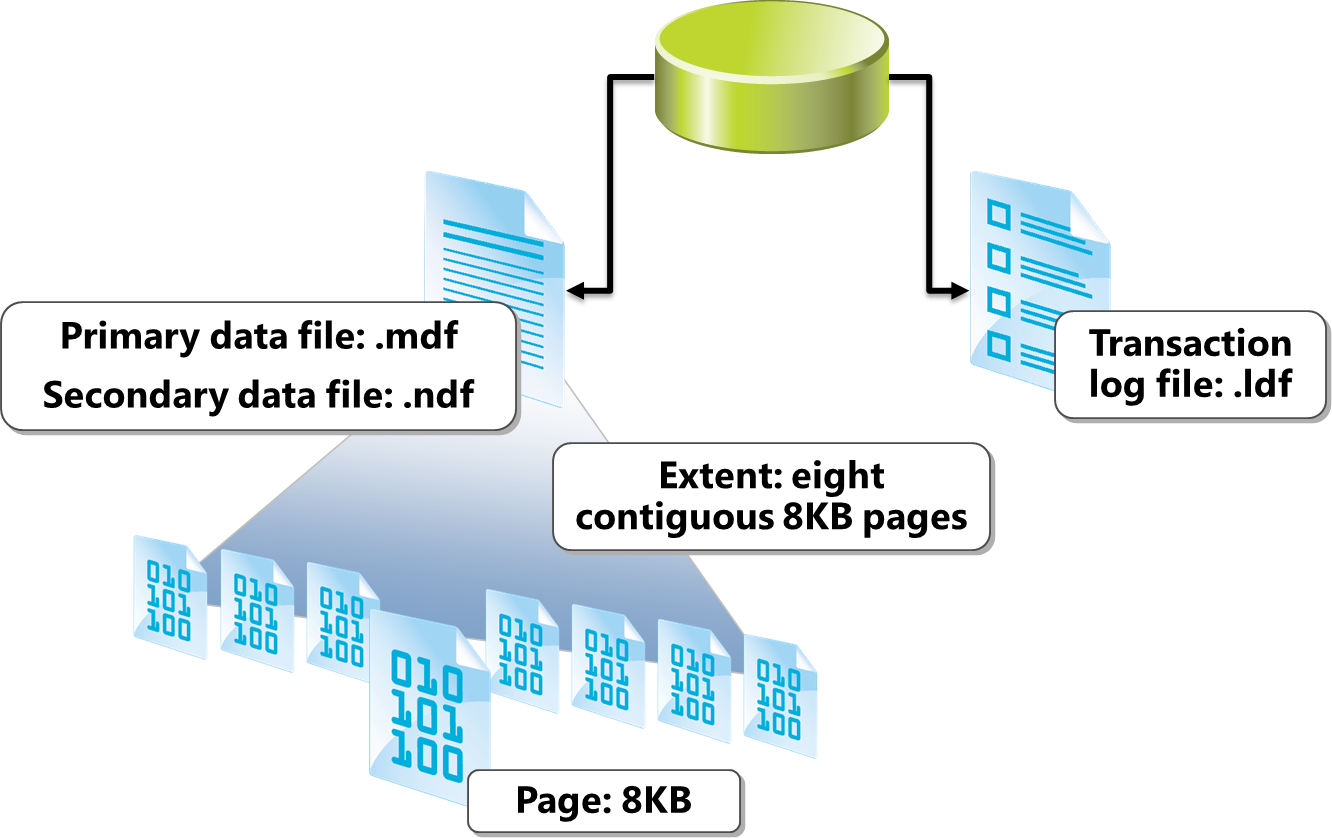
**Обязанности и задачи DBA**

* **Установка** и **настройка** сервера, создание баз данных и распределение их информации и файлов журналов на устройствах хранения.
* Администратор должен обеспечивать **безопасность** базы данных с тем, чтобы осуществлялась **проверка подлинности** и **авторизация** пользователей. Он назначает **роли** сервера и базы данных, регулирует **доступ** пользователей к информации.
* После того, как база данных создана и заполнена данными требуется ее постоянное **обслуживание** и **оптимизация**. Это предполагает уменьшение фрагментации, которая появляется по мере того, как записи добавляются и удаляются, а также сохранение файлов.
* Базы данных часто имеют решающее значение для деловых операций, поэтому главной задачей DBA является планирование стратегии **резервного копирования** и **восстановления** базы.
* **Импорт** и **экспорт** данных. Данные часто передаются между системами, поэтому администраторам баз необходимо уметь выполнять экспорт или импорт данных.
* **Мониторинг** и **устранение неполадок** систем баз данных. Многие операции по администрированию баз данных предполагают принятие мер для устранения неполадок и возникающих проблем. Администраторы БД должны осуществлять упреждающий подход, чтобы попытаться обнаружить потенциальные проблемы до того, как они начнут влиять на операции с данными.
* **Документирование** процедур управлениябазой данных.Администраторы БД обычно составляют и ведут документацию, 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. Особенно в случае, когда новый администратор должен взять на себя ответственность за управление базой данных, или при возникновении чрезвычайной ситуации в отсутствии администратора.

**Хранение данных в MS SQL Server**

Существует **три** типа файлов базы данных SQL Server: первичный файл данных, вторичные файлы данных и файлы журналов транзакций.



**Первичный файл данных** вкаждой базе данных присутствует обязательно. Кроме собственно страниц данных, первичный файл содержит указатели на другие файлы базы. Первичные файлы данных обычно используют расширение **.mdf**.

**Вторичные файлы данных** являются необязательными, определяются пользователем как дополнительные. Вторичные файлы можно использовать для распределения данных по нескольким дискам. 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** содержат сведения, которые можно использовать для восстановления базы данных. Все транзакции записываются в файл журнала для обеспечения целостности данных в случае сбоя и для поддержки отката транзакций. Рекомендуемое расширение для файлов журнала – **.ldf**.

**Страницы и экстенты файлов данных.** Файлы данных хранят информацию в виде страниц, которые сгруппированы в **экстенты** – группы из **восьми** непрерывных страниц.

Дисковые операции ввода-вывода выполняются на уровне страницы. Каждая страница имеет размер **8** КБ.

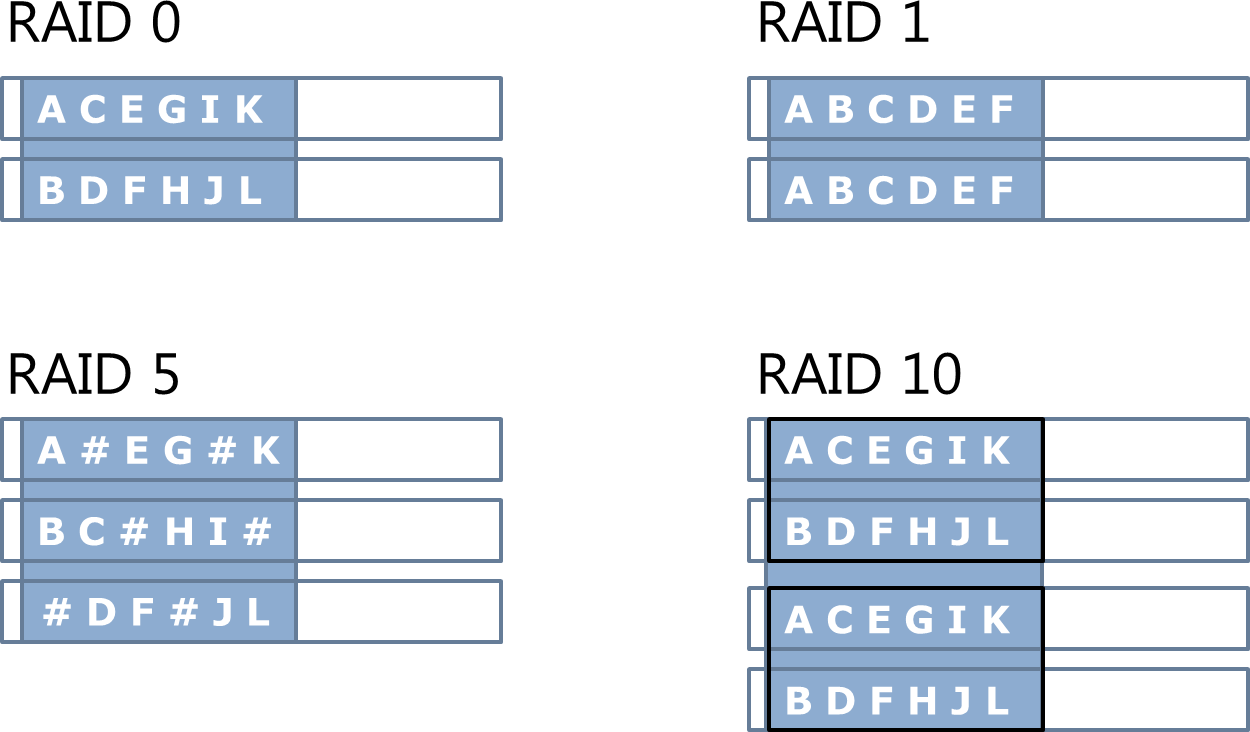
Страницы в файле данных нумеруются последовательно, начиная с нуля. Каждый файл в базе данных также имеет свой уникальный идентификационный номер.

SQL Server использует экстенты для упрощения управления памятью в файлах данных.

**Уровни RAID**. Многие решения для ***хранения данных*** используют ***аппаратное*** обеспечение RAID для реализации отказоустойчивости и повышения производительности.

**RAID** − это технология, с помощью которой может быть организовано хранение данных на жёстких дисках, т.е. это **инструкция**, как хранить и читать файлы на дисках (***Redundant Array of Independent Disks*** − массив независимых дисков с избыточностью).

Часто используемые типы 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. Уровень RAID 1 обеспечивает отказоустойчивость и, в целом, улучшает производительность считывания, но может снизить производительность записи.
  + **RAID 5** (чередование с четностью). 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. RAID 10 предлагает наилучшее сочетание производительности чтения/записи и отказоустойчивости, но является самым дорогостоящим решением.

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.

**Системные базы данных**

SQL Server использует системные базы данных для поддержания внутренних метаданных. Экземпляр SQL Server всегда содержит пять системных баз данных – ***master***, ***msdb***, ***model***, ***tempdb***, и ***resource***.

|  |  |
| --- | --- |
| **System database** | **Описание** |
| **master** | Хранит все системные настройки. Если база *master* повреждена, SQL Server не запускается, поэтому важно делать резервную копию *master*. |
| **msdb** | Содержит сведения о задачах обслуживания баз данных. Важно регулярно создавать резервные копии этой базы, чтобы не потерять истории для резервного копирования, восстановления и планы обслуживания. |
| **model** | Любая новая база данных использует *model* в качестве шаблона. Если создать объекты в базе *model*, они будут присутствовать во всех новых базах данных на экземпляре сервера. SQL Server не запустится, если *model* отсутствует. |
| **tempdb** | Содержит временные данные. |
| **resource** | Скрытая база данных только для чтения, содержит системные объекты, которые отображаются в схеме ***sys*** каждой базы данных. Также содержит все системные хранимые процедуры, системные представления и системные функции. |

Все системные базы данных, за исключением базы данных ***resource***, можно **переместить** на новые места, чтобы **сбалансировать нагрузку** ввода/вывода. Но нужно 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 значение для общей производительности SQL Server.

Большинство объектов в базе создаются пользователями и представляют собой временные таблицы, табличные переменные и т. п. 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.

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

**Администрирование базы данных MS SQL Server**

### Большинство административных задач можно выполнять как с помощью **графического пользовательского интерфейса** в среде SQL Server Management Studio (SSMS), так и на **языке Transact-SQL**.

### 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).

*Графические интерфейсы в SSMS имеют кнопку Script, с помощью которой автоматически генерируется эквивалентный код T-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 базы данных или для удаления.
* **Системные хранимые процедуры и функции**. SQL Server предоставляет системные хранимые процедуры и функции, которые обеспечивают решение задач настройки и управления системой. 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.

На **начальном** этапе после установки сервера необходимо выбрать режим **аутенти**фикации, то есть обеспечить возможность ***соединения*** пользователя с сервером.

Затем надо **авторизовать** пользователей, чтобы обеспечить ***доступ к ресурсам***. При этом определяется, какие ресурсы может использовать пользователь.

Пользователю нужны права на выполнение определенных действий. Обычно для этой цели создаются пользователи или **роли** баз данных и им предоставляются **разрешения** (иначе привилегии). Если нужно предоставить пользователю права на уровне всего сервера, а не отдельной базы данных, можно воспользоваться серверными ролями.

**Аутентификация и создание учетной записи**

**Аутентификация** — это процесс входа в SQL Server, когда пользователь отправляет свои данные на сервер. Аутентификация устанавливает личность пользователя.

MS SQL Server использует два режима аутентификации (проверки подлинности): аутентификацию Windows и аутентификацию MS SQL Server.

|  |  |
| --- | --- |
| Для проверки подлинности по умолчанию используется ***аутентификация Windows*** – это режим, который использует для подключения к **серверу** только логины Windows.  В этом случае пользователю нет необходимости вводить пароли при подключении к SQL Server, если он уже вошел в Windows. |  |
| Для использования ***аутентификации SQL Server*** необходимо создать учетную запись в самом SQL Server.  Логины SQL Server – это самостоятельные учетные записи со своими именами и паролями, информация о которых хранится в системной базе данных master. |  |

Предпочтительный вариант для пользователя − это ***логин Windows***. При этом пользователю достаточно помнить пароль для входа на свой компьютер. Также снижается размер системных таблиц базы данных master, в результате чего аутентификация производится быстрее.

Использование ***логинов SQL Server*** может быть обусловлено тем, что часто на предприятиях администрированием SQL Server и домена Windows занимаются разные люди, которым сложно согласовывать свои действия.

Логины SQL Server позволяют администратору базы данных быть независимым от администратора домена. Кроме того, на предприятии может не оказаться домена Windows (если сеть построена на основе NetWare или UNIX) или пользователи могут не входить в домен (если они подключаются из филиалов или через Web-интерфейс с домашнего компьютера).

Логины Windows − это удобство и защищенность, логины SQL Server, − это гибкость и независимость от администратора сети.

Таким образом, режим аутентификации может быть осуществлен либо с помощью проверки подлинности Windows, либо в **смешанном** режиме, когда выполняется и аутентификация Windows и аутентификация SQL Server.

**Создание учетной записи администратора**

Для создания **учетной записи** в SQL Server надо соединится с сервером как администратор Windows.

В SQL Server Management Studio список учетных записей, сконфигурированных на сервере, находится в **Безопасность** / **Имена входа**.

Скрипт:

**USE [master]**

**GO**

**CREATE LOGIN [student] WITH PASSWORD=N'studentfit',**

**DEFAULT\_DATABASE=[master], DEFAULT\_LANGUAGE=[русский],**

**CHECK\_EXPIRATION=ON, CHECK\_POLICY=ON**

**GO**

**ALTER SERVER ROLE [sysadmin] ADD MEMBER [student]**

Проверка срока действия, проверка подлинности

**Создание и хранение баз данных**

Создавать базы данных может только **пользователь с правами системного администратора** с помощью графического интерфейса в среде SSMS или командой **CREATE DATABASE**.

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).

В следующем примере 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.

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

Существует общий принцип: не стоит обращаться из клиентского приложения к таблицам базы данных напрямую. Для *изменения* данных лучше использовать ***хранимые процедуры***, а для запросов на чтение − ***хранимые процедуры*** или ***представления***.

Если потребуется поменять структуру базы данных (например, какую-то таблицу поделить на текущую и архивную или добавить новый столбец) не нужно будет вносить изменения в клиентское приложение.

**Создание пользователя базы данных**

Предоставив доступ к экземпляру SQL Server, необходимо предоставить доступ к определенным базам данных.

Доступ к базам данных предоставляется посредством добавления **пользователей** (или иначе **принципалов**) базы данных и сопоставления пользователям базы данных **имен входа**.

**Создание учетной записи (имени входа) нового пользователя**

**Пользователь** – это участник базы данных, привязанный к определенному имени входа.

**USE [master]**

**GO**

**CREATE LOGIN [Director] WITH PASSWORD=N'store',**

**DEFAULT\_DATABASE=[master], DEFAULT\_LANGUAGE=[русский],**

**CHECK\_EXPIRATION=ON, CHECK\_POLICY=ON**

**Создание нового пользователя и сопоставление с именем входа**

**конкретной базы данных**

Пользователи базы данных создаются на уровне**базы данных** и используются для предоставления разрешений в базе данных (на таблицы, представления, хранимые процедуры).

**USE [ПРОДАЖИ]**

**GO**

**CREATE USER [Director] FOR LOGIN [Director]**

**GO**

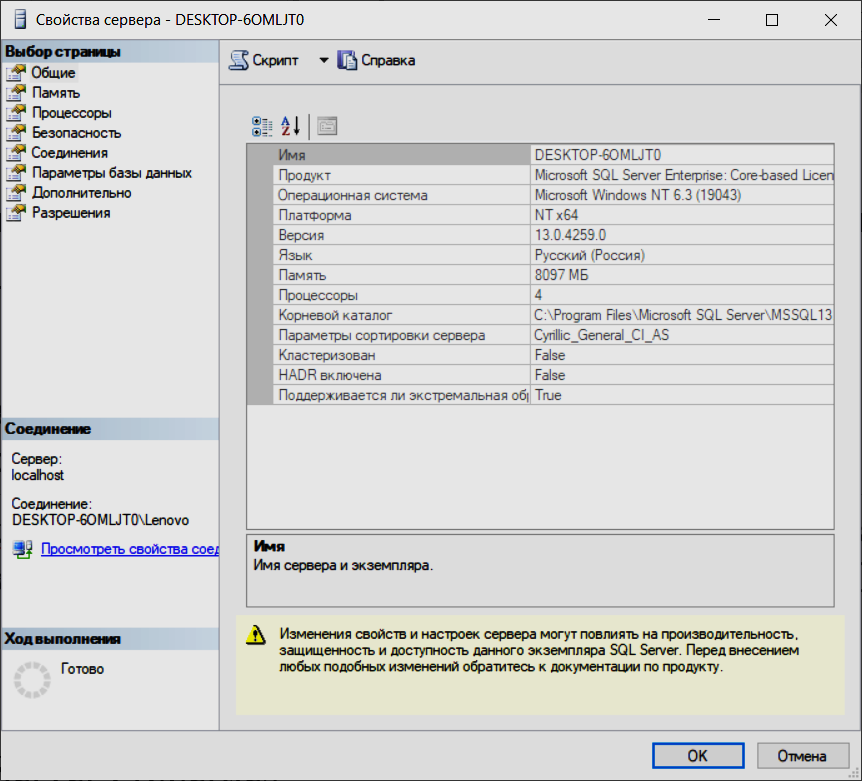
**USE [ПРОДАЖИ]**

**ALTER ROLE [db\_owner] ADD MEMBER [Director]**

**Соединение нового пользователя с базой данных**

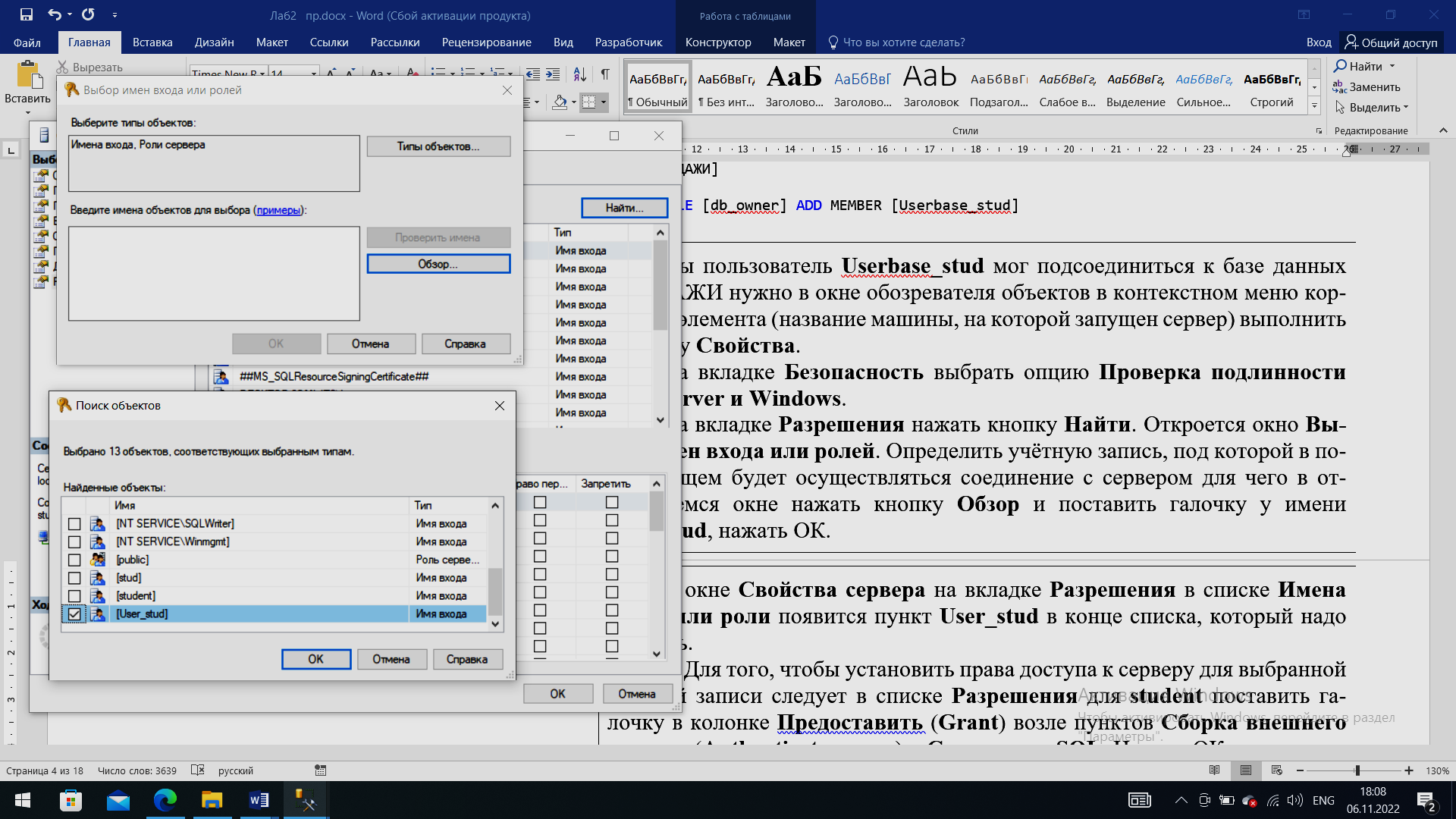
Чтобы пользователь **Director** мог подсоединиться к созданной базе данных следует выполнить несколько подготовительных действий.

1. В окне Object Explorer выбрать корневой элемент (название машины, на которой запущен сервер) и в контекстном меню выполнить команду **Свойства**. Откроется окно свойств сервера.



2. На вкладке **Безопасность** выбрать опцию **Проверка подлинности SQL Server и Windows**.

3. На вкладке **Разрешения** нажать кнопку **Найти**. Откроется окно **Выбор имен входа или ролей**.



Определить учётную запись, под которой в последующем будет осуществляться соединение с сервером для чего в открывшемся окне нажать кнопку **Обзор** и поставить галочку у имени **Director**. Нажать ОК, произойдет возврат в окно свойств.

В конце списке **Имена входа или роли** появится пункт **Director**, который надо выбрать. В списке **Разрешения** **для** **Director** поставить галочку в колонке **Предоставить** (**Grant**) возле пунктов **Администрировать …** и **Соединение SQL**. Нажать ОК.

4. Перезагрузить сервер для чего в контекстном меню корневого элемента выполнить команду **Перезапустить**.

**use [master]**

**GO**

**GRANT ADMINISTER BULK OPERATIONS TO [Director]**

**GRANT CONNECT SQL TO [Director]**

Удалить пользователя из базы данных, например, ПРОДАЖИ можно при помощи инструкции DROP USER.

**USE [ПРОДАЖИ]**

**DROP USER [Director]**

**Параметры базы данных**

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.

**Параметры автоматические** (**Auto options**). Контролируют некоторые автоматические модели поведения. Как правило, в большинстве систем рекомендуется отключить параметры Auto Close (автоматическое закрытие) и Auto Shrink (автоматическое сжатие), но включить автоматическое создание и обновление статистики (Auto Create Statistics и Auto Update Statistics).

**Параметры курсора** (**Cursor options**). 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**). 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** в установки параметров. 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.

# **Резервное копирование и восстановление**

# **баз данных и журналов**

Одним из наиболее важных аспектов является обеспечение регулярного копирования данных, чтобы в случае сбоя можно было восстановить БД.

Существует 2 режима создания резервных копий.

**Статическое резервное копирование** – режим, при котором только одна активная сессия, которая создает резервную копию. Другие пользовательские процессы во время выполнения копирования недопустимы.

**Динамическое резервное копирование** – режим, при котором копирование может выполняться без остановки сервера баз данных, удаления пользователей или даже закрытия файлов.

**Типы резервных копий**

**Полная** (**Full**) копия – в полную резервную копию включаются ***все*** объекты, системные таблицы и данные, а также фрагменты журналов транзакций, соответствующие времени архивации. Позволяет полностью восстановить базу.

**Разностная** (**Differential**) копия – предназначена для архивации данных, которые изменились со времени ***последнего*** резервного копирования. Сохраняются только *изменения*, поэтому архивация занимает меньше времени.

Копия **журнала транзакций** (**Transaction Log**) – сохраняются ***изменения***, произошедшие со времени последнего резервного копирования ***журнала транзакций***, а затем выполняется его усечение, в ходе которого стираются выполненные или отмененные транзакции.

Копия **файлов** и **файловых групп** (**File/File Group**) – позволяют архивировать не всю БД, а только *указанные* файлы и файловые группы.

Одновременно 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.

**Частичная** (**Partial**) копия – содержит данные только из некоторых ***файловых групп***, 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.

Копия **заключительного фрагмента журнала** (**Tail-log Backup**) – в резервную копию заключительного фрагмента журнала попадают все записи, резервная копия которых еще не была создана, что позволяет предотвратить потерю работы и сохранить неповрежденную цепочку журналов.

Для восстановления базы данных 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**) – иногда приходится выполнять резервное копирование БД для особых нужд, когда это не сказывается на общем процессе резервного копирования и восстановления.

**Полная резервная копия**

Скрипт, например, для базы данных ПРОДАЖИ:

**BACKUP DATABASE ПРОДАЖИ TO**

**DISK = 'D:\БД\Backup\ПРОДАЖИ\_РЕЗЕРВ.bak'**

**WITH NOFORMAT, INIT,**

**NAME = N'ПРОДАЖИ-Полная База данных Резервное копирование',**

**SKIP, NOREWIND, NOUNLOAD, STATS = 10**

Обработано 448 страниц для базы данных "ПРОДАЖИ", файл "ПРОДАЖИ" для файла 1.

Обработано 2 страниц для базы данных "ПРОДАЖИ", файл "ПРОДАЖИ\_log" для файла 1.

BACKUP DATABASE успешно обработал 450 страниц за 0.483 секунд (7.278 MБ/сек).

**Дифференциальная (разностная) резервная копия**

**базы данных**

Если полная резервная копия базы данных имеется, но ее создание занимает большой промежуток времени, то можно не создавать заново полную копию, а использовать разностную копию БД.

При этом периодически следует делать полную копию БД, чтобы уменьшить объемы создаваемых разностных копий (после создания полной копии БД, все предыдущие разностные копии теряют свою актуальность).

Например, можно выполнять полное резервное копирование всей базы данных один раз в неделю, а затем в течение недели регулярно создавать разностные резервные копии.

Скрипт для базы данных ПРОДАЖИ:

**BACKUP DATABASE ПРОДАЖИ TO**

**DISK = N'D:\БД\Backup\ПРОДАЖИ\_РЕЗЕРВ.bak'**

**WITH DIFFERENTIAL, NOFORMAT, NOINIT,**

**NAME = N'ПРОДАЖИ-Полная База данных Резервное копирование',**

**SKIP, NOREWIND, NOUNLOAD, STATS = 10**

Обработано 40 страниц для базы данных "ПРОДАЖИ", файл "ПРОДАЖИ" для файла 2.

Обработано 2 страниц для базы данных "ПРОДАЖИ", файл "ПРОДАЖИ\_log" для файла 2.

BACKUP DATABASE WITH DIFFERENTIAL успешно обработал 42 страниц за 0.146 секунд (2.247 MБ/сек).

**Резервная копия журнала транзакций**

Скрипт для базы данных ПРОДАЖИ:

**BACKUP LOG ПРОДАЖИ TO**

**DISK = N'D:\БД\Backup\ПРОДАЖИ\_12\_12\_2022.bak'**

**WITH NOFORMAT, INIT,**

**NAME = N'ПРОДАЖИ-Полная База данных Резервное копирование',**

**SKIP, NOREWIND, NOUNLOAD, STATS = 10**

Пример кода SQL для создания резервной **разностной** копии БД с ***проверкой*** копии по завершению (отличается от полного копирования флагом DIFFERENTIAL вместо него нужно использовать NOFORMAT).

**declare @pathBackup as varchar(55)**

**set @pathBackup = N'D:\Backup\ПРОДАЖИ\_' +**

**REPLACE(convert(varchar,GETDATE(), 104),'.','\_') + '.bak'**

**BACKUP DATABASE ПРОДАЖИ TO DISK = @pathBackup**

**WITH DIFFERENTIAL, NOFORMAT, INIT,**

**NAME = N'Полная База данных Резервное копирование',**

**SKIP, NOREWIND, NOUNLOAD, STATS = 10, CHECKSUM**

**GO**

**--проверка копии**

**declare @backupSetId as int**

**declare @pathBackup as varchar(55)**

**set @pathBackup = N'D:\Backup\ПРОДАЖИ\_' +**

**REPLACE(convert(varchar,GETDATE(), 104),'.','\_') + '.bak'**

**select @backupSetId = position from msdb..backupset**

**where database\_name=N'ПРОДАЖИ'**

**and backup\_set\_id=(select max(backup\_set\_id) from msdb..backupset**

**where database\_name=N'ПРОДАЖИ')**

**if @backupSetId is null**

**begin**

**raiserror(N'Ошибка верификации. Сведения о резервном копировании для базы данных "ПРОДАЖИ" не найдены.', 16, 1)**

**end**

**RESTORE VERIFYONLY FROM DISK = @pathBackup**

**WITH FILE = @backupSetId, NOUNLOAD, NOREWIND**

**GO**

### **Параметры для обеспечения целостности резервной копии**

Резервное копирование SQL Server включает параметры для обеспечения целостности резервных копий.

**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 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 BaseOne TO**

**DISK = 'R:\Backups\BO.bak'**

**MIRROR TO DISK = 'Q:\Backups\BO\_M.bak'**

**WITH FORMAT, INIT;**

**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 BaseOne TO**

**DISK = 'R:\Backups\BO.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\BO.bak'**

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

**Создание плана резервного копирования**

**базы данных**

План создается с учетом следующих критериев:

* Допустимый объём потерянных данных (за последний день/час/минуту);
* Требования к дисковому пространству и его стоимость;
* Затраты ресурсов сервера на резервное копирование.

Рассмотрим настройку резервного копирования с помощью графического интерфейса SSMS по плану:

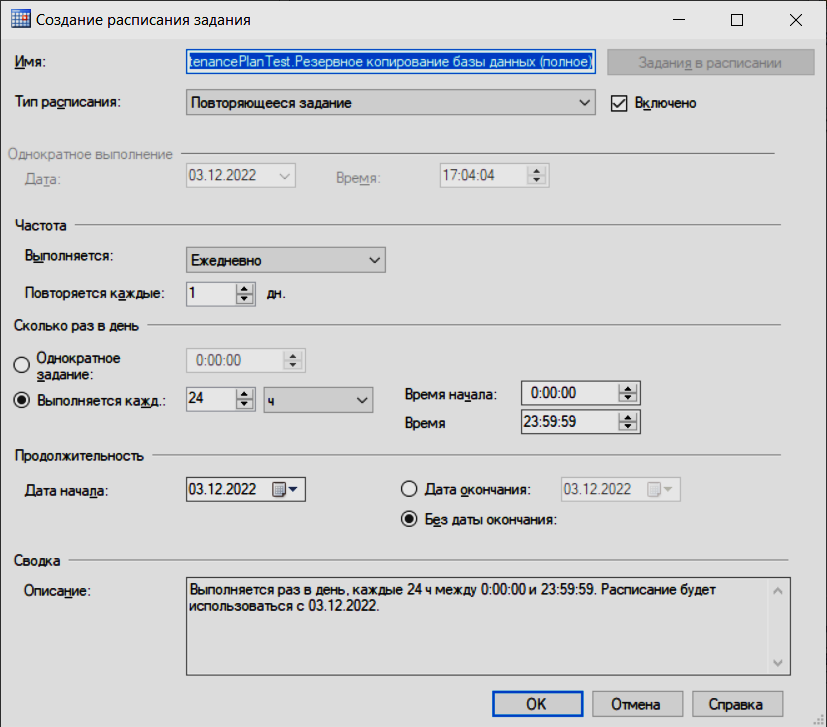
* *Полная* резервная копия каждые 24 часа
* Копия *журнала транзакций* – каждые 30 минут

В SSMS перейти в раздел **Управление**. В контекстном меню пункта **Планы обслуживания** выполнить команду **Мастер планов обслуживания**. В появившемся окне ввести *имя плана*, выбрать режим «*Отдельное расписание для каждой задачи*», нажать **Далее**.

Выбрать операции, которые нужно сделать в этом плане обслуживания: «*Резервное копирование базы данных (полное)*» и «*Резервное копирование базы данных (журнал транзакций)*».

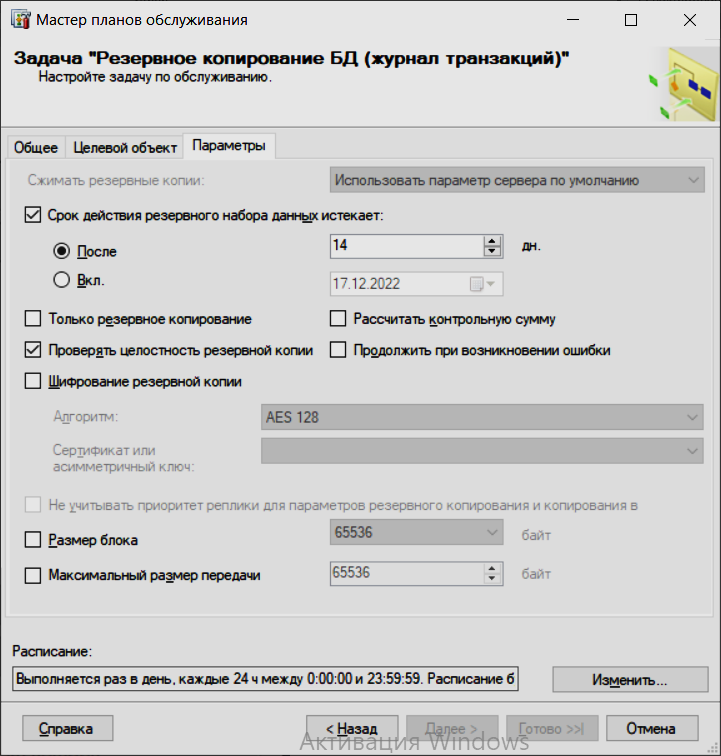
Оставить последовательность операций в этом порядке.

Выбрать базу данных SQL Server, которую нужно резервировать и, нажав кнопку **Изменить** определить расписание.

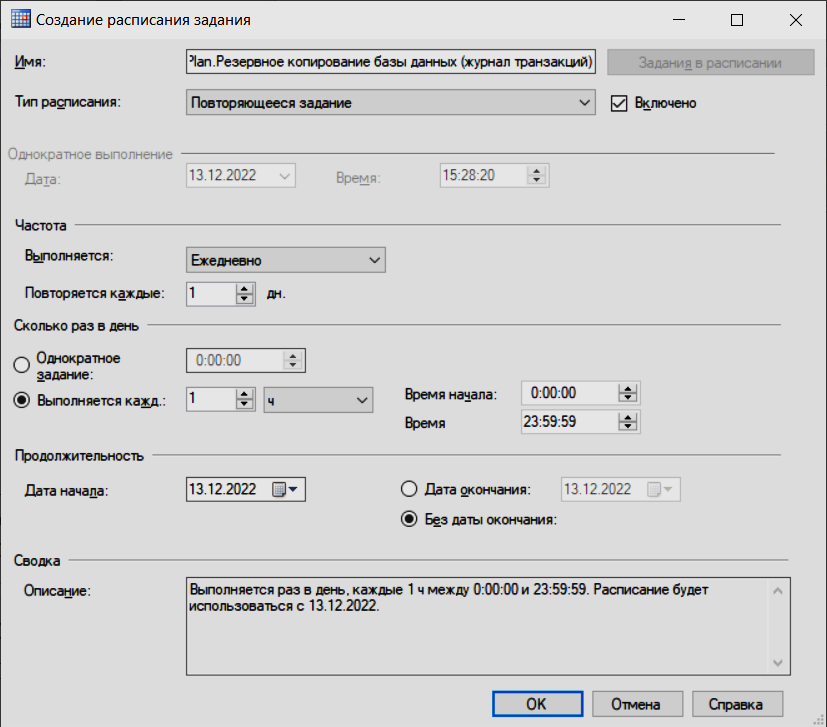


Нажать **Ок** и перейти на вкладку **Целевой объект**, где следует указать путь к каталогу (отметив пункт **Создать резервную копию баз данных в одном или нескольких файлах** и нажав кнопку **Добавить**), в который нужно сохранять резервные копии базы данных.

На вкладке **Параметры** указать, сколько будут храниться резервные копии (например, 14 дней).



Нажать **Далее** и аналогично создать расписание резервного копирования для *журнала транзакций*.



Выполнить план обслуживания вручную и проверить журнал. На этом настройка резервного копирования закончена.

Для **удаления плана** в разделе **Управление** надо раскрыть пункт **Планы обслуживания** и в контекстном меню имени плана выполнить команду **Удалить**.

Либо можно узнать о наличии плана, выполнив команду:

**select \* from msdb.dbo.sysmaintplan\_plans**

Определить из результата **id** плана и удалить план с помощью:

**delete from msdb.dbo.sysmaintplan\_subplans where plan\_id = ''**

**delete from msdb.dbo.sysmaintplan\_plans where id = ''**

# **Рекомендации по резервному копированию SQL Server**

* **Резервные копии не должны храниться на том же диске, что и SQL Server**. При выходе из строя основного дискового массива важно иметь доступ к резервным копиям.
* **Процесс резервного копирования должен минимально влиять на работу пользователей**. Полные резервные копии лучше делать тогда, когда пользовательская активность на сервере минимальна.
* **Регулярно следует проверять целостность резервных копий и проводить тестовые восстановления**.
* **Заранее следует рассчитать время, необходимое для полного восстановления при аварии**. Часто в базах хранится критически важная для бизнеса информация, поэтому руководитель должен знать минимальное время, которое потребуется для восстановления после аварии.

### **Сжатие резервных копий**

Резервные файлы могут быстро стать очень большими, поэтому SQL Server позволяет сжимать их. При этом не допускается одновременное существование сжатых и несжатых резервных копий.

Для сжатия резервной копии, можно использовать параметр WITH COMPRESSION в инструкции BACKUP:

**BACKUP DATABASE BaseOne TO**

**DISK = 'R:\Backups\BO\_Comp.bak'**

**WITH COMPRESSION**

Если настройка резервных копий по умолчанию со сжатием, но надо это отменить, то следует использовать опцию NO\_COMPRESSION.

Коэффициент сжатия резервной копии можно вычислить, 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;**

Сжатие существенно повышает нагрузку на центральный процессор, что может помешать выполнению других операций.

### **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 надо использовать, и сертификат (или асимметричный ключ), который может использоваться для шифрования данных.

Чтобы использовать шифрование резервных копий нужно:

1. Создать 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. Создать сертификат или асимметричный ключ для шифрования резервной копии. Можно создать сертификат или асимметричный ключ с помощью оператора CREATE CERTIFICATE или CREATE ASYMMETRIC KEY.
3. Выполнить 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 можно было восстановить базу данных на другой экземпляр SQL Server в случае полного отказа сервера.

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

**BACKUP DATABASE BaseOne TO DISK = 'R:\Backups\BO\_Encrypt.bak'**

**WITH FORMAT, INIT, ENCRYPTION(ALGORITHM=AES\_128,**

**SERVER CERTIFICATE = [BackupCert])**

## Восстановление баз данных

Используются различные типы моделей восстановления.

**Полная** (**Full**) модель – 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**) – модель сокращает пространство, занимаемое журналом транзакций, сохраняя б**о**льшую 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 операции и массовые загрузки придется повторить вручную. Стратегия резервного копирования для данной модели должна включать те же архивы, что и для полной модели.

**Простая** (**Simple**) модель – предназначена для восстановления до точки последней архивации. Включив простую модель восстановления, нельзя выполнять резервное копирование журналов транзакций.

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

**Копирование данных**. Стадия копирования данных является, как правило, самой длинной в восстановлении базы данных. Процесс восстановления считывает заголовок резервной копии, и SQL Server восстанавливает требуемые файлы данных и журналов. Затем 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**. SQL Server извлекает данные из журнала транзакций.

### В простой модели восстановления эти данные извлекаются либо из полной резервной копии, либо из разностной. Эти сведения дополняются содержимым резервных копий журнала транзакций, т.е. «накатываются» все изменения, которые содержатся в деталях журнала транзакций вплоть до точки восстановления.

### **Стадия отмены.** 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 база данных переводится 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.

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

##### **Восстановление данных из полной резервной копии**

##### **при помощи T-SQL**

Чтобы восстановить базу данных ПРОДАЖИ используется инструкция RESTORE DATABASE. При выполнении запросов с базой данных не должно быть установлено ни одного соединения.

**USE master**

**GO**

**--Переход в однопользовательский режим**

**ALTER DATABASE ПРОДАЖИ SET SINGLE\_USER WITH ROLLBACK IMMEDIATE**

**--Восстановление данных из полной резервной копии**

**RESTORE DATABASE ПРОДАЖИ FROM DISK=N'D:\БД\BACKUP\ПРОДАЖИ\_РЕЗЕРВ.BAK'**

В процессе этой процедуры восстанавливаются только данные полной резервной копии. После восстановления база данных автоматически переводится в рабочий режим, возвращаясь к исходным параметрам.

##### **Восстановление данных из полной и разностной резервных копий при помощи T-SQL**

Если нужно также применить разностную резервную копию, необходимо сообщить SQL Server о том, что не следует переводить базу данных в рабочий режим до завершения полного восстановления, поскольку это сделает невозможным восстановление данных из разностной резервной копии. Для этого используется параметр NORECOVERY.

Параметр NORECOVERY должен присутствовать во всех инструкциях RESTORE, за исключением последней в цикле восстановления.

**USE master**

**GO**

**RESTORE DATABASE ПРОДАЖИ FROM**

**DISK=N'D:\БД\BACKUP\ПРОДАЖИ\_РЕЗЕРВ.BAK'**

**WITH NORECOVERY;**

**RESTORE DATABASE ПРОДАЖИ FROM**

**DISK=N'D:\БД\BACKUP\ПРОДАЖИ\_РЕЗЕРВ.BAK';**

###### **Восстановление данных из полной резервной копии**

###### **и резервных копий журнала транзакций**

Пример. Для создания резервных копий базы данных ПРОДАЖИ используется скрипт:

**USE master;**

**ALTER DATABASE ПРОДАЖИ SET RECOVERY FULL;**

**--Создание полной резервной копии базы данных**

**BACKUP DATABASE ПРОДАЖИ TO**

**DISK = N'D:\БД\BACKUP\ПРОДАЖИ\_РЕЗЕРВ.BAK'**

**WITH INIT;**

**--Имитирование выполнения транзакции**

**UPDATE ПРОДАЖИ.dbo.Заказы SET Цена\_продажи = 80**

**WHERE Наименование\_товара='Стол';**

**--Создание резервной копии журнала транзакций**

**BACKUP LOG ПРОДАЖИ TO DISK = N'D:\БД\BACKUP\ПРОДАЖИ\_LOG1.BAK'**

**WITH INIT;**

**--Имитирование выполнения транзакции**

**UPDATE ПРОДАЖИ.dbo.Заказы SET Цена\_продажи = 100**

**WHERE Наименование\_товара='Стол';**

Предположим, что файл базы данных ПРОДАЖИ поврежден. Можно создать резервную копию журнала транзакций, чтобы зафиксировать последние записи, которые содержат транзакции, завершенные после создания последней резервной копии журнала транзакций. Для этого следует воспользоваться особым параметром NO\_TRUNCATE.

**--Выполнение резервного копирования последних записей журнала транзакций**

**BACKUP LOG ПРОДАЖИ TO DISK = N'D:\БД\BACKUP\ПРОДАЖИ\_LOG2.BAK'**

**WITH INIT, NO\_TRUNCATE;**

Если журнал транзакции поврежден, дальнейшее выполнение этого типа резервного копирования будет невозможным. Можно восстановить данные только из резервной копии, созданной ранее.

После того, как резервная копия журнала будет создана, можно использовать инструкцию RESTORE, чтобы сначала выполнить восстановление данных из полной резервной копии, а затем из двух резервных копий журнала транзакций.

Команда RESTORE включает параметры WITH RECOVERY и WITH NORECOVERY.

Опция WITH RECOVERY является параметром по умолчанию и может не указываться. Опция гарантирует, что база данных переводится в оперативный режим сразу после восстановления.

Если стратегия требует, чтобы после полной резервной копии базы данных были восстановлены дополнительные резервные копии, то все операции восстановления должны быть с опцией NORECOVERY за исключением последней.

Для всех операций восстановления за исключением последней используется параметр NORECOVERY.

**--Переключение на базу данных master**

**USE master**

**GO**

**--Переход в однопользовательский режим**

**ALTER DATABASE ПРОДАЖИ SET SINGLE\_USER WITH ROLLBACK IMMEDIATE**

**--Восстанавление данных из полной резервной копии**

**RESTORE DATABASE ПРОДАЖИ FROM**

**DISK=N'D:\БД\BACKUP\ПРОДАЖИ\_РЕЗЕРВ.BAK'**

**WITH REPLACE, NORECOVERY;**

**--Восстанавление данных из первой резервной копии журнала транзакций**

**RESTORE LOG ПРОДАЖИ FROM**

**DISK = N'D:\БД\BACKUP\ПРОДАЖИ\_LOG1.BAK'**

**WITH NORECOVERY;**

**--Восстанавление данных из второй резервной копии журнала транзакций**

**RESTORE LOG ПРОДАЖИ FROM**

**DISK = N'D:\БД\BACKUP\ПРОДАЖИ\_LOG2.BAK';**

Параметр REPLACE указывает на то, что следует пропустить проверку безопасности и заменить файлы базы данных без дальнейших запросов.

**SELECT Цена\_продажи FROM ПРОДАЖИ.dbo.Заказы**

**WHERE Наименование\_товара='Стол';**

**--Переход в многопользовательский режим**

**ALTER DATABASE ПРОДАЖИ SET MULTI\_USER**

### **Причины потери данных**

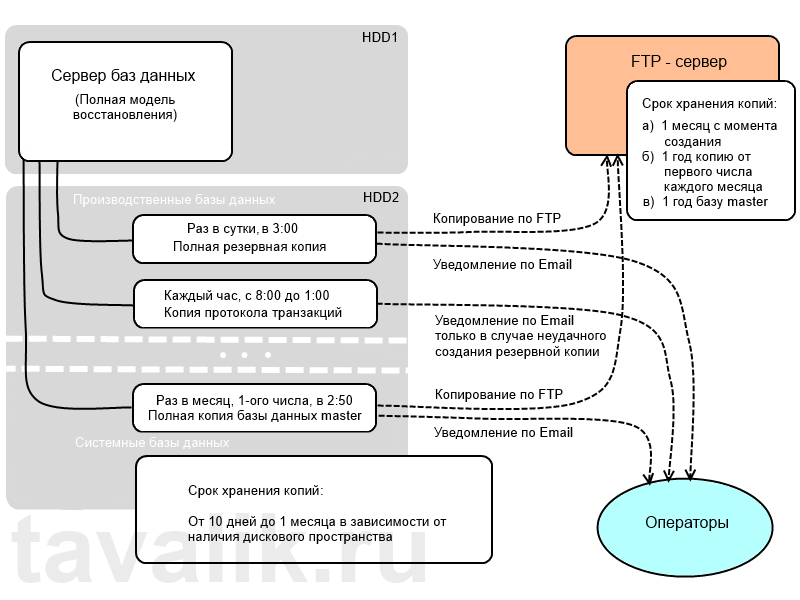
1. **Программные ошибки**. Поскольку такие ошибки основываются на дефектах программной логики, система баз данных не может выполнить восстановление в подобных ситуациях. Поэтому восстановление должен проводить сам программист, выполнив обработку таких исключений.
2. **Ошибки администратора (человеческий фактор).** Случаи, в которых пользователь с большими полномочиями может неумышленно (или умышленно) разрушить данные. Необходимо постараться создать такой режим работы, который сделает подобную ситуацию маловероятной, однако совсем исключать такую возможность нельзя.
3. **Выход из строя компьютера (сбой системы)** возникает в результате ошибок в оборудовании и программном обеспечении. В этом случае содержимое оперативной памяти компьютера может быть потерянно. В качестве защиты можно рекомендовать использование резервного сервера, зеркальное отображение баз данных и пр.
4. **Отказ дискового накопителя, например, ф**изическое разрушение жесткого диска. Рекомендуется использование технологий RAID для хранения файлов баз данных, кроме того необходимо, чтобы файлы резервных копий хранились на дисковом носителе, отличном от устройства, на котором располагаются файлы баз данных.
5. **Катастрофы (пожар, наводнение, землетрясение) или кража** — Обойти эту ситуацию станет возможным, если устройство, содержащее необходимую для восстановления данных информацию, будет храниться отдельно от основного оборудования и не будет потеряно в результате катастроф или краж.

### **Какие базы данных и как часто копировать?**

1. База данных **master**является наиболее важной базой данных системы, потому что она содержит информацию обо всех базах данных в этой системе. Поэтому резервное копирование базы данных master должно происходить на регулярной основе. Кроме того, рекомендуется создавать копию каждый раз, когда выполняются действия, приводящие к модификации базы данных master. Вот некоторые из них:
   * Выполнение операторов и хранимых процедур;
   * Создание, изменение и удаление базы данных;
   * Изменения протокола транзакций.
2. Следует выполнять резервное копирование всех ***производственных***баз данных на регулярной основе. Дополнительно, необходимо делать резервную копию после того как с базами данных были выполнены следующие изменения:
   * После создания базы данных;
   * После создания индексов;
   * После создания протокола транзакций;
   * После выполнения непротоколируемых операций (операции, которые не записываются в протокол транзакций).

### Пример стратегии резервного копирования

Ниже приводится некоторый план резервного копирования

[](http://tavalik.ru/wp-content/uploads/2013/06/Plan_rezerv_kopirovaniya_01.png)

Некоторые характеристики, связанные с выполнением резервного копирования базы данных **DB1:**

Аппаратная часть:

* + ОЗУ 24 Гб
  + HDD 250 Гб RAID 1+0

Программное обеспечение:

* + Microsoft SQL Server 2016

Базы данных:

* + Общий объем баз данных: ~ 95 Гб
  + Объем базы данных **master**: ~ 5 Мб
  + Объем базы данных **DB1**: ~ 23 Гб
  + Модель восстановления базы данных **DB1**: Полная
  + Прирост базы данных **DB1** в день: ~ 200 Мб

Резервные копии базы данных **DB1**:

* + Время создания полной резервной копии: ~ 5 мин.
  + Время создания копии протокола транзакций: ~ 4 сек.
  + Размер полной резервной копии (с сжатием): ~ 1,6 Гб
  + Размер копии протокола транзакций: ~ 20 Мб

Необходимое дисковое пространство для плана резервного копирования **DB1**:

* + Хранение полных резервных копий (1 месяц): ~ 50 Гб
  + Хранение копий протокола транзакций (1 месяц): ~ 10 Гб
  + Хранение полных копий от 1-ого числа каждого месяца (1 год): ~ 20 Гб
  + Итого размер дискового пространства: ~ 80 Гб
  + Итого размер дискового пространства в % от размера базы: ~ 350 %

Время восстановления базы данных **DB1**:

* + Время восстановления полной резервной копии: ~ 5 мин.
  + Время восстановления копии протокола транзакций: ~ 5 сек.

**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 этом нужно использовать логическое имя файла, которое задается файлу при создании базы данных.

Узнать логическое имя файла можно с помощью системного представления *sys.database\_files*.

Пример. Перевести базу данных **BaseOne** в автономный режим, переместить файл данных и вернуть БД в режим online:

**ALTER DATABASE BaseOne SET OFFLINE;**

**MODIFY FILE (NAME = BODataFile, FILENAME = 'C:\BODataFile.mdf');**

**ALTER DATABASE BaseOne SET ONLINE;**

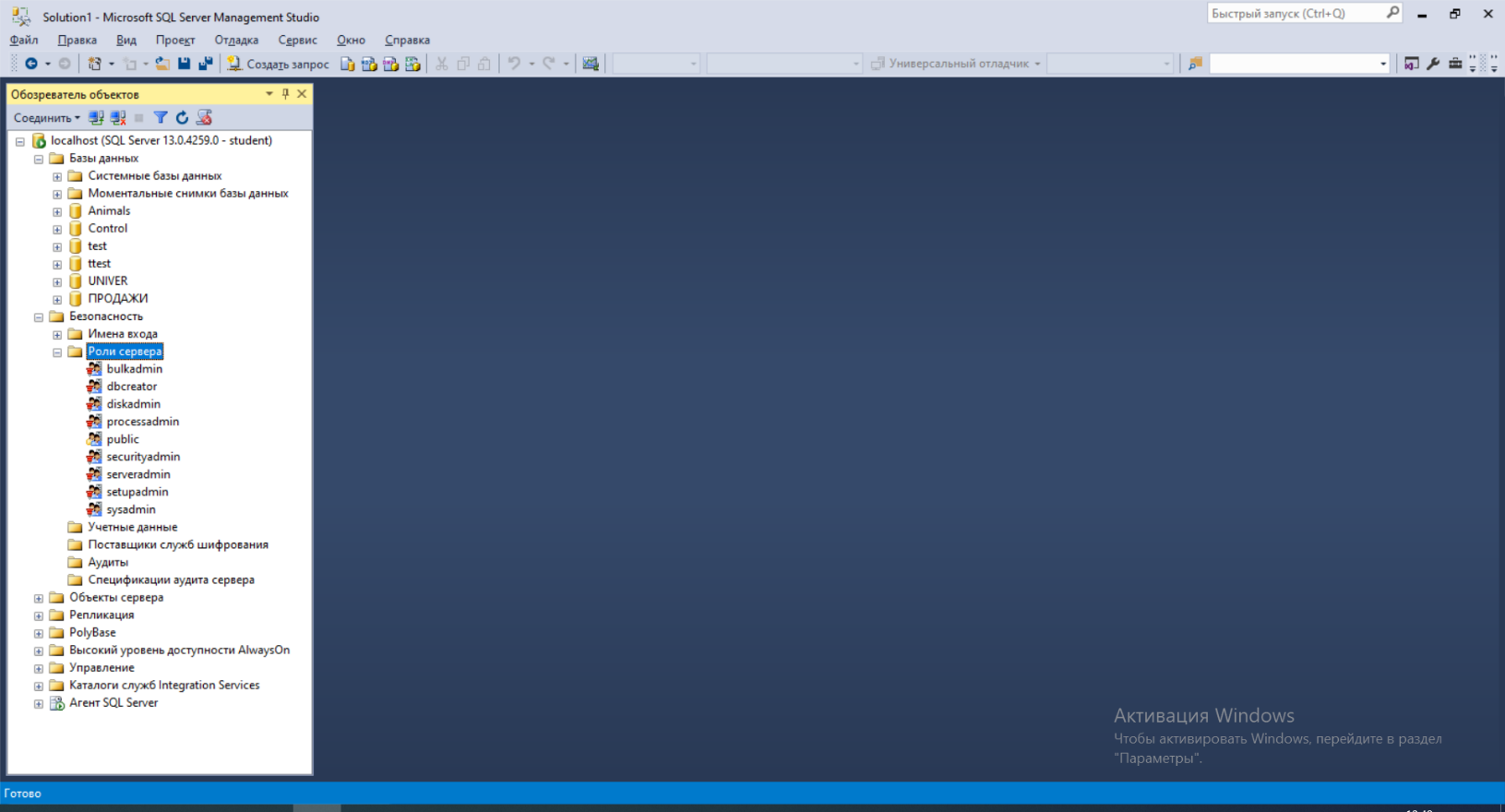
**Авторизация в SQL Server**

**Авторизация** – это процесс определения того, к каким защищаемым объектам может обращаться пользователь и какие операции ему **разрешены**.

Для авторизации SQL Server использует безопасность на основе ролей.

**Роль** − это именованный набор *разрешений* (или иначе *привилегий*) на выполнение различных операций.

В SQL Server есть 3 уровня безопасности, их можно представить, как иерархию от высшего к низшему:

* **Уровень сервера** – на этом уровне можно раздать права на базы данных, учетные записи, роли сервера и группы доступности;
* **Уровень базы данных** включают в себя схемы, пользователи базы данных, роли базы данных и полнотекстовые каталоги;
* **Уровень схемы** включают такие объекты, как таблицы, представления, функции и хранимые процедуры.

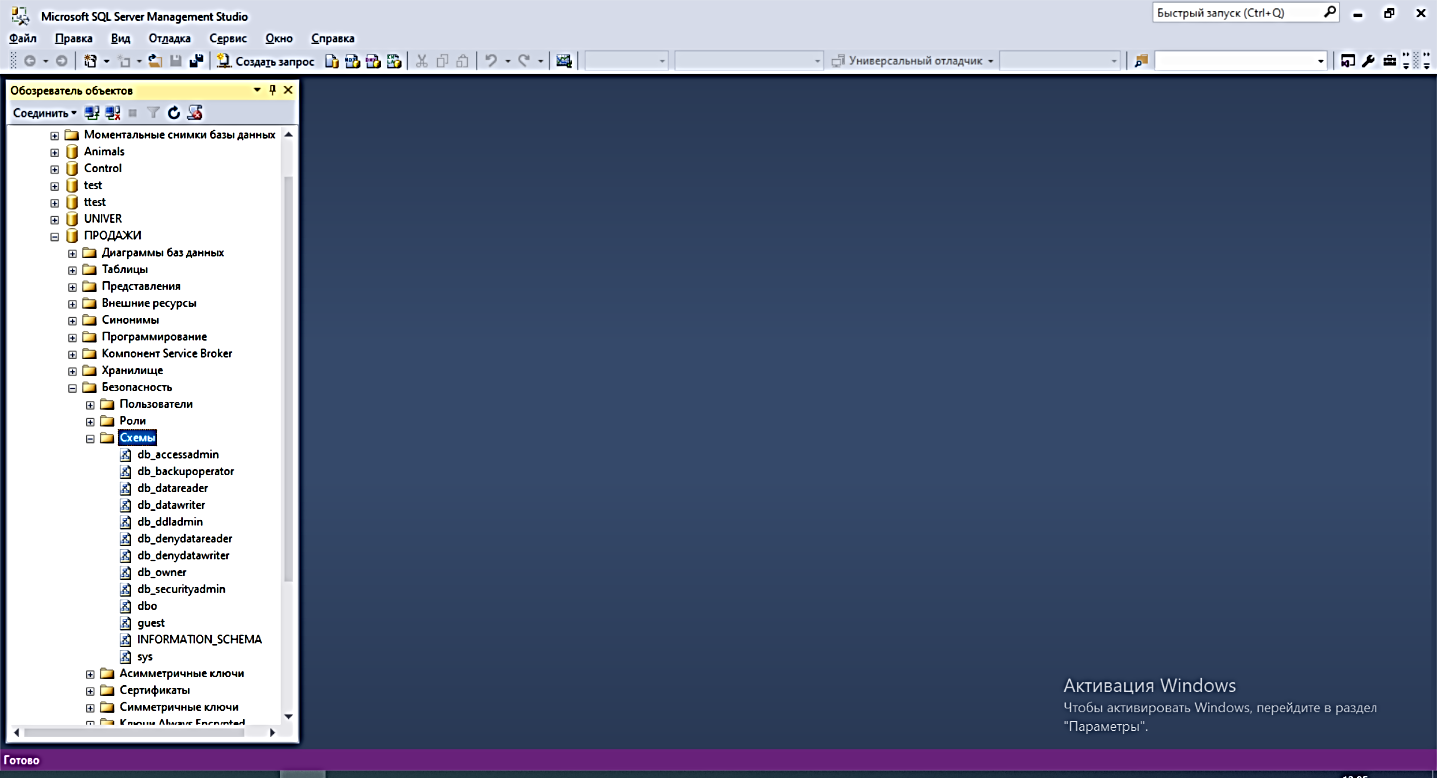
**Встроенные роли сервера**

Список ролей можно увидеть в пункте **Безопасность** / **Роли сервера**.

|  |  |
| --- | --- |
| **Роль** | **Описание** |
| **sysadmin** | Участник роли имеет полные права ко всем ресурсам SQL Server. |
| **serveradmin** | Можно изменять параметры конфигурации на уровне сервера и выключать сервер. |
| **securityadmin** | Участники роли могут предоставлять права доступа с помощью команд GRANT, DENY и REVOKE на уровне сервера и на уровне базы данных, если имеют к ней  доступ (securityadmin мало чем отличается от роли sysadmin, участники этой роли потенциально могут получить доступ ко всем ресурсам). |
| **rocessadmin** | Можно завершать процессы, запущенные в SQL Server. |
| **setupadmin** | Можно добавлять и удалять связанные серверы с помощью T-SQL. |
| **bulkadmin** | Участники роли могут запускать BULK INSERT операции. |
| **diskadmin** | Можно управлять устройствами резервного копирования (почти не применяется). |
| **dbcreator** | Можно создавать, изменять, удалять и восстанавливать базы данных. |
| **public** | Каждый логин SQL Server находится в этой роли. Изменить членство public нельзя |

# **Схемы баз данных**

**Схема** формально определяется как набор объектов в базе данных.

К схеме могут относиться таблицы, представления, хранимые процедуры, пользовательские функции, ограничения целостности, пользовательские типы данных и другие объекты, на которые приходится предоставлять разрешения чаще всего.

Список схем можно увидеть в пункте **Безопасность** / **Схемы** конкретной базы данных.

При создании любой базы данных в ней автоматически создаются четыре специальных пользователя:

**dbo** (от database owner) – пользователь-владелец базы данных. Он автоматически создается для того логина, от имени которого была создана эта база данных. Как владелец, он получает полные права на свою базу данных (при помощи встроенной роли базы данных db\_owner);

**guest** (гость) − этот пользователь предназначен для предоставления разрешений всем логинам, которым не соответствует ни один пользователь в базе данных. По умолчанию у этого пользователя нет права login для базы данных, и, следовательно, работать он не будет. Этот пользователь используется чаще всего для предоставления разрешений логинам на какие-то учебные/тестовые базы данных или на базы данных-справочники, доступные только на чтение;

**INFORMATION\_SCHEMA** − этому пользователю не может соответствовать ни один логин. Его единственное значение − быть владельцем схемы этой схемы, в которой хранятся представления системной информации для базы данных;

**sys** − этому специальному пользователю не могут соответствовать логины. Он является владельцем схемы sys, которой принадлежат системные объекты базы данных.

**Роли баз данных**

Различают ***системные*** ***роли*** и ***пользовательские роли*** баз данных.

***Системные роли базы данных***

**public** − эта специальная роль предназначена для предоставления разрешений сразу всем пользователям базы данных. Специально сделать пользователя членом этой роли или лишить его членства невозможно. Все пользователи базы данных получают права этой роли автоматически.

**db\_owner** − этой роли автоматически предоставляются полные права на базу данных. Изначально права этой роли предоставляются только специальному пользователю dbo, а через него − логину, который создал эту базу данных;

**db\_accessadmin** − роль для сотрудника, ответственного за пользователей базы данных. Этот сотрудник получит возможность создавать, изменять и удалять объекты пользователей баз данных, а также создавать схемы. Других прав у него нет;

**db\_securityadmin** − эта роль дополняет роль db\_accessadmin. Сотрудник с правами этой роли получает возможность назначать разрешения на объекты базы данных и изменять членство во встроенных и пользовательских ролях;

**db\_backupoperator** − эта роль дает право выполнять резервное копирование базы данных;

**db\_ddladmin** − эта роль применяется в редких ситуациях, когда пользователю необходимо дать право создавать, изменять и удалять любые объекты в базе данных, не предоставляя прав на информацию, которая содержится в существующих объектах;

**db\_datareader** и **db\_datawriter** − эти встроенные роли дают право просматривать и изменять (в том числе добавлять и удалять) любую информацию в базе данных.

Часто пользователю необходимо дать права на чтение и запись информации во всех таблицах базы данных, не предоставляя ему лишних административных разрешений (на создание и удаление объектов, изменение прав и т. п.). Самый простой вариант в этой ситуации − воспользоваться этими двумя ролями.

**db\_denydatareader** и **db\_denydatawriter** − эти роли запрещают явно просматривать какие−либо данные и запрещают внесение изменений.

Явный запрет всегда имеет приоритет перед явно предоставленными разрешениями. Обычно эти роли используются в ситуации, когда "разрешаем всем, а потом некоторым запрещаем".

***Пользовательские роли базы данных***

При создании новых пользователей, рекомендуется использовать принцип LUA (Least-privileged User Account или Аккаунт с **Наименьшими** Правами). Этот принцип является важной частью безопасности сервера и данных.

Для пользователей с административными правами рекомендуется выдавать разрешения только на те операции, которые им будут необходимы.

Когда пользователей становится много, управлять их разрешениями становится сложнее, и также становится сложнее не допустить ошибки в предоставлении прав.

Рекомендуется предоставлять разрешения **ролям**, а пользователей добавлять в роли.

Чтобы для базы данных создать пользовательскую роль нужно подключиться к серверу под именем системного администратора, в обозревателе объектов выбрать базу, в контекстном меню пункта **Безопасность** этой базы выполнить **Создать** / **Роль базы данных**. Ввести имя роли и нажать кнопку **Добавить**. В появившемся окне выбрать тип объекта **Пользователи**, ввести имя пользователя,нажать **Ок**.

Работа с разрешениями может производится тремя способами для всех объектов базы:

− на вкладке **Разрешения** свойств этого объекта (эта вкладка предусмотрена не для всех объектов, для которых можно предоставить разрешения);

− на странице **Защищаемые объекты** окна свойств пользователя или роли;

− при помощи команд **GRANT** (предоставить разрешение), **DENY** (явно запретить что−то делать) и **REVOKE** (отменить явно предоставленное разрешение или запрет).

Рассмотрим ключевые слова, которые могут использоваться в данных командах.

ALTER − возможность вносить любые изменения в свойства объекта (за исключением смены владельца).

CONTROL − тот, кому предоставлено такое разрешение, получает полные права как на сам объект, так и на информацию в нем.

DELETE − возможность удалять существующую информацию в таблицах. Применяется к таблицам, представлениям и столбцам.

EXECUTE − право запускать на выполнение. Применяется к хранимым процедурам и функциям.

INSERT − право на вставку новых данных в таблицы. Применяется к таблицам, представлениям и столбцам.

REFERENCES − разрешение, которое можно предоставить для проверки ограничений целостности.

Например, пользователь может добавлять данные в таблицу с внешним ключом, а на таблицу с первичным ключом ему нельзя предоставлять права на просмотр. В этом случае на таблицу с первичным ключом ему можно предоставить право REFERENCES − и он сможет производить вставку данных в таблицу с внешним ключом, не получая лишних прав на главную таблицу. Это разрешение можно предоставлять на таблицы, представления, столбцы и функции.

SELECT − право на чтение информации. Предоставляется для таблиц, представлений, столбцов и табличных функций.

TAKE OWNERSHIP − право на принятие на себя владения данным объектом. Владелец автоматически обладает полными правами на свой объект. Такое право можно назначить для любых объектов.

UPDATE − возможность вносить изменения в существующие записи в таблице. Предоставляется на таблицы, представления и столбцы.

VIEW DEFINITION − право на просмотр определения для данного объекта. Предусмотрено для таблиц, представлений, процедур и функций.

Пример. Пусть имеется пользователь **Seller** с именем учетной записи **Seller**.

Скрипт:

**USE [ПРОДАЖИ]**

**GO**

**CREATE ROLE [RoleSeller]**

**GO**

**USE [ПРОДАЖИ]**

**GO**

**ALTER ROLE [RoleSeller] ADD MEMBER [Seller]**

Запрет *изменений* в таблице **Заказчики** и разрешение изменений в таблице **Заказы** можно осуществить с помощью скрипта:

**use [ПРОДАЖИ]**

**GO**

**DENY ALTER ON [dbo].[Заказчики] TO [RoleSeller]**

**GRANT ALTER ON [dbo].[Заказы] TO [RoleSeller]**

С помощью следующей команды пользователю **Seller** базы данных ПРОДАЖИ предоставляются права выборки и изменения данных таблицы **Заказы** этой базы данных:

**GRANT SELECT, UPDATE ON ПРОДАЖИ.Заказы.Наименование\_товара TO Seller**

**Использование схем и представлений**

В большинстве реальных задач используются десятки и даже сотни таблиц и других объектов базы данных. Предоставлять каждому пользователю разрешения на каждый из этих объектов очень неудобно. Целесообразно использовать разрешения на ***уровне схемы*** или всей базы данных.

Если назначить пользователю разрешения на схему, то он получит разрешения на все объекты этой схемы.

Применение схемы дает ряд преимуществ:

− нескольким пользователям можно назначить одну и ту же схему по умолчанию, что может быть удобно при разработке приложений;

− несколько пользователей могут владеть одной и той же схемой, при этом один пользователь может являться владельцем сразу нескольких схем;

− при удалении пользователя не придется переименовывать его объекты;

− упрощается предоставление разрешений для наборов объектов в базе данных.

Пример. Пусть требуется для базы данных ПРОДАЖИ создать схему **Расчеты** с владельцем **Director**.

**USE ПРОДАЖИ;**

**GO**

**CREATE SCHEMA Расчеты AUTHORIZATION Director;**

Создание таблицы **Счета** в схеме **Расчеты**:

**CREATE TABLE Расчеты.Счета**

**( IDсчета int,**

**Датасчета smalldatetime,**

**IDклиента int**

**);**

Предоставление разрешения SELECT на таблицу **Счета** роли *public*.

**GRANT SELECT ON Расчеты.Счета TO public;**

Добавление строки данных в новую таблицу:

**INSERT INTO Расчеты.Счета VALUES (101,getdate(),102);**

В SQL Server не допускается удаление схемы, если в схеме есть объекты. Получение информации о схемах:

**SELECT \* FROM sys.schemas;**

Получение информации о схеме **Расчеты** для проверки, есть ли в схеме объекты.

**USE ПРОДАЖИ;**

**GO**

**SELECT s.name AS "Расчеты",o.name AS "Object" FROM**

**sys.schemas s INNER JOIN sys.objects o**

**ON s.schema\_id=o.schema\_id WHERE s.name='Расчеты';**

Удаление таблицы **Счета** из схемы **Расчеты**:

**DROP TABLE Расчеты.Счета;**

Удаление схемы **Расчеты**:

**DROP SCHEMA Расчеты;**

На практике лучше ***не пользоваться разрешениями на уровне отдельных столбцов*** из-за падения производительности и усложнения системы разрешений. Если пользователю можно видеть не все столбцы в таблице (например, ему не нужны домашние телефоны сотрудников), то правильнее будет создать ***представление*** или ***хранимую процедуру***, которые будут отфильтровывать ненужные столбцы.

Если пользователю можно видеть не все столбцы в таблице, то правильнее будет создать **представление**.

Для создания представления пользователь должен иметь привилегию SELECT во всех таблицах, на которые он ссылается в представлении.

Пример. Пусть надо дать пользователю **Seller** возможность видеть только столбцы **Наименование\_товара** и **Цена** в таблице **Товары** базы данных ПРОДАЖИ. Для этого можно использовать представление:

**CREATE VIEW Sellerview**

**AS SELECT Наименование\_товара, Цена FROM ПРОДАЖИ;**

Затем следует предоставить **Seller** разрешение SELECT в представлении:

**GRANT SELECT On Sellerview to Seller;**

### **Оператор GRANT**

Предоставляет разрешения на выполнение действий на защищаемом объекте участникам. 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.

**GRANT SELECT, INSERT ON Заказы TO User1, User2;**

Все привилегии объекта используют *один и тот же синтаксис*, кроме команд UPDATE и REFERENCES, в которых не обязательно указывать имена столбцов.

**GRANT UPDATE ON Заказы TO User3;**

Эта команда позволит **User3** изменять значения в любом или во всех столбцах таблицы **Заказы**.

Однако, если надо ограничить **User3** и позволить вносить изменения только в столбцах **Товар** и **Количество**, то можно использовать команду:

**GRANT UPDATE (Товар, Количество) ON Заказы TO User3;**

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).

### **Оператор DENY**

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:

**DENY INSERT ON Заказы TO User3;**

*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:

**REVOKE UPDATE ON Заказы TO User3;**

Оператор REVOKE удаляет только явные разрешения, его нельзя использовать для переопределения наследуемых разрешений.

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.

Пример.

Роль **Role1**получает разрешения SELECT для доступа к таблице **Заказы**:

**GRANT SELECT ON OBJECT::Заказы TO Role1;**

**User4**является членом роли **Role1**. Кроме того, ему предоставлено разрешение по его собственному имени пользователя посредством инструкции:

**GRANT SELECT ON OBJECT:: Заказы TO User4;**

Предположим, администратор хочет удалить разрешение GRANT для роли **Role1**. Если администратор правильно выполняет инструкцию:

**REVOKE SELECT ON OBJECT:: Заказы TO Role1;**

то пользователь **User4** не утрачивает разрешение SELECT к таблице **Заказы**.

Если администратор неправильно удаляет разрешение:

**DENY SELECT ON OBJECT:: Заказы TO Role1;**

то пользователь **User4**, как член роли **Role1**, утрачивает разрешение SELECT.

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

* В среде SSMS на вкладке **Разрешения** диалогового окна свойств защищаемого объекта или на вкладке **Защищаемые объекты** диалогового окна свойств участника. Здесь можно выбрать сочетание участника и защищаемого объекта, которое надо 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 сделав запрос к системной функции *sys.fn\_my\_permissions*, указав защищаемый объект, для которого требуется просмотреть действующие разрешения.

В примере показано, как выполнить просмотр действующих разрешений для имени входа **Student1**в таблице **dbo.Товары**:

**EXECUTE AS LOGIN = 'Student1'**

**SELECT \* FROM sys.fn\_my\_permissions ('dbo.Товары', 'Object');**

**REVERT**

Инструкция REVERT возвращает выполнение в контекст участника, вызывавшего последнюю инструкцию EXECUTE AS.

**Схемы** — это коллекции объектов (таблиц, представлений, хранимых процедур и т. д.).

**Роли** — коллекции прав: роли серверов для прав на уровне сервера/экземпляра и роли базы данных для прав в конкретной базе данных.

Независимость пользователей баз данных от схем предоставляет администраторам и разработчикам некоторые преимущества:

• одной схемой могут владеть несколько пользователей-членов ролей или групп Windows, что расширяет функциональность, позволяя ролям и группам владеть объектами:

• упрощается управление пользователями баз данных. Удаление пользователей не влияет на объекты, содержащиеся в схеме этого пользователя, поэтому не нужно проверять и тестировать приложения, явно ссылающиеся на содержащиеся в схеме объекты после удаления пользователя, создавшего эти объекты;

• несколько пользователей могут делить одну схему по умолчанию, создавая единое пространство имен;

• общие схемы по умолчанию позволяют разработчикам хранить общие объекты в схеме, специально созданной для данного приложения, а не в схеме DBO;

Полные имена объектов при использовании схем состоят из четырех компонентов: ***server.database.schema.object***.

Если приложение ссылается на объект базы данных, не уточняя схему, сервер пытается найти его в схеме по умолчанию для текущего пользователя. При неудачном поиске сервер делает попытку поиска в схеме ***dbo***.