

Microsoft SQL Server

Программирование
и администрирование СУБД

Урок №6

Безопасность

Содержание

1. Системные таблицы и представления	4
2. Защита данных. Режимы защиты данных. Понятие аутентификации и авторизации на уровне MS SQL Server	8
2.1. Основные понятия	8
2.2. Создание имен входа	11
2.3. Модификация и удаление имен входа	23
3. Пользователи базы данных	26
4. Роли	32
4.1. Понятие ролей. Фиксированные роли	32
4.2. Пользовательские роли	41

5. Управление правами доступа	47
5.1. Основные понятия	47
5.2. Объектные права доступа	48
5.3. Командные права доступа	60
5.4. Данные про права	64

1. Системные таблицы и представления

Важно знать, что каждая база данных, кроме созданных пользователем таблиц, содержит дополнительный набор системных таблиц. Имена большинства системных таблиц начинаются с префикса `sys`. Большинство полей этих таблиц документированы, но случаются и не документированные поля, обращение к которым осуществлять запрещено.

Итак, давайте разберемся, какие системные таблицы существуют и какая информация в них представлена. Все системные таблицы можно разделить на следующие группы:

1. Таблицы резервного копирования и восстановления.
2. Таблицы доставки журналов. Их имена начинаются с префикса `log_shipping_xxx`.
3. Таблицы системы отслеживания изменений данных.
4. Таблицы репликации.
5. Таблицы плана обслуживания базы данных.
6. Таблицы агента SQL Server.
7. Таблицы служб Integration Services, которые фактически содержат описание системных таблиц базы данных `msdb`.
8. Базовые системные таблицы, которые сохраняют метаданные отдельной базы данных.

Итак, набор системных таблиц довольно большой. Основное внимание привлекают базовые системные таблицы. Они размещаются в базах данных пользователей и в главной базе данных master. Но данные базовых системных таблиц закрыты и недоступны для обычных клиентов SQL Server. Доступ к ним имеют только сотрудники корпорации Microsoft.

Существует еще одна база данных, которая содержит перечень всех системных объектов, которые входят в состав SQL Server, то есть включают также в себя набор базовых системных таблиц пользовательских баз данных. Это база данных Resource. Системные объекты SQL Server, такие как sys.objects, физически хранятся в базе данных Resource, а отражаются в схеме sys для каждой базы данных (поскольку сама база данных Resource невидимая и доступ напрямую к ней запрещен).

Стоит отметить, что ряд системных таблиц, которые присутствовали в предыдущих версиях SQL Server, начиная с версии MS SQL Server 2005 реализованы в виде набора системных представлений. Эти системные представления несут общую информацию и реализованы на серверном уровне, то есть могут рассказать о любой базе данных на сервере. Ряд этих представлений вы уже знаете, поскольку о них мы упоминали в предыдущих уроках. Но несмотря на это, подытожим всю известную информацию и приведем сопоставление системных таблиц SQL Server 2000 (последняя версия, где использовались приведенные системные таблицы) и системных представлений SQL Server 2008.

Сначала рассмотрим системные представления, которые есть в каждой базе данных на сервере.

Системная таблица SQL Server 2000	Системное представление SQL Server 2008	Описание
sysobjects	sys.objects	Содержит перечень объектов базы данных, кроме триггеров.
syscomments	sys.sql_modules	Содержит код объявления объектов базы данных (модулей) и различные параметры, с которыми создавался каждый отдельный объект.
syscolumns	sys.columns	Содержит общее описание каждого поля таблицы, представления и возвращаемое табличное значение функции.
sysdepends	sys.sql_expression_dependencies	Содержит данные про именуемые зависимости между представлениями, хранимыми процедурами, триггерами и таблицами, которые указываются при их объявлении.
sysfilegroups	sys.filegroups	Содержит информацию о файловых группах, которые есть в базе данных.
sysfiles	sys.database_files	Содержит данные о файлах базы данных.
sysforeignkeys	sys.foreign_key_columns	Сохраняет информацию обо всех внешних ключах, включая данные о том, с первичным полем которой таблицы он связан.
sysconstraints	sys.check_constraints sys.default_constraints sys.key_constraints sys.foreign_keys	Содержит информацию о конструкции базы данных: ограничение (check), значение по умолчанию (default), ключи (key), включая внешние (foreign keys).

Системная таблица SQL Server 2000	Системное представление SQL Server 2008	Описание
sysindexes	sys.indexes sys.partitions sys. allocation_units sys.dm_db_ partition_stats	Содержат информацию обо всех индексах базы данных.
sysindexkeys	sys. index_columns	Содержит информацию об индексах и полях, на которые они указывают.
sysmembers	sys.database_ role_members	Содержит информацию о пользовате- лях и роли, к которым они относятся.
sysreferences	sys.foreign_keys	Содержит карту связей таблиц.
systypes	sys.types	Содержит список всех пользователь- ских и системных типов данных.
sysusers	sys.database_ principals	Содержит данные обо всех пользова- телях и роли SQL Server и Windows.
sysfulltext- catalogs	sys. fulltext_catalogs	Содержит список полно- текстовых каталогов.
syspermissions sysprotects	sys.database_ permissions sys.server_ permissions	Содержит данные обо всех пра- вах и ограничениях отдельно- го пользователя или роли: на уровне поля базы данных (sys. database_permissions) или на уров- не сервера (sys.server_permissions).

Ряд системных таблиц базы данных master в SQL Server 2008 также были заменены на системные представления с аналогичными именами. Например, системная таблица sysconfigures, которая содержит параметры конфигурации сервера в системе, заменена на системное представление sys.configurations, системная таблица sysdatabases с информацией о перечне баз данных сервера – на системное представление sys.databases и тому подобное.

2. Защита данных. Режимы защиты данных. Понятие аутентификации и авторизации на уровне MS SQL Server

2.1. Основные понятия

Сегодня мы научимся правильно организовывать безопасность данных в SQL Server 2008, которая имеет важное значение в деятельности разного рода организаций, использующих базы данных. Ведь без обеспечения надежной защиты данных, любой может уничтожить ценные данные или получить доступ к секретной информации. Для этого в первую очередь следует правильно спроектировать систему управления пользователями, в частности уменьшить количество пользователей с неограниченным доступом к минимуму, ввести систему паролей и т.п.

Система безопасности SQL Server 2008 для предотвращения несанкционированного доступа к данным использует режимы защиты и учетные записи. Существует 2 режима защиты данных:

1. Режим защиты Windows (Windows authentication) – это самый простой способ защиты и он является режимом защиты данных по умолчанию. Этот режим

аутентификации позволяет подключаться к SQL Server, вводя логин и пароль при входе в Windows. Чтобы задействовать этот режим, достаточно на сервере добавить имя входа Windows.

2. Смешанный режим защиты (Mixed mode) – это режим защиты, при котором пользователи регистрируются непосредственно при доступе к SQL Server, отдельно от регистрации в операционной системе. При этом доступ к данным могут получать любые имена входа, включая имена входа Windows. Обычно, данный способ аутентификации выбирают в случае использования отличной от Windows операционной системы.

Корпорация Microsoft рекомендует использовать режим защиты Windows, поскольку он позволяет реализовать преимущества всех централизованных политик безопасности домена Active Directory. Такой режим действительно имеет ряд преимуществ для производственной базы данных, среди которых можно выделить следующие:

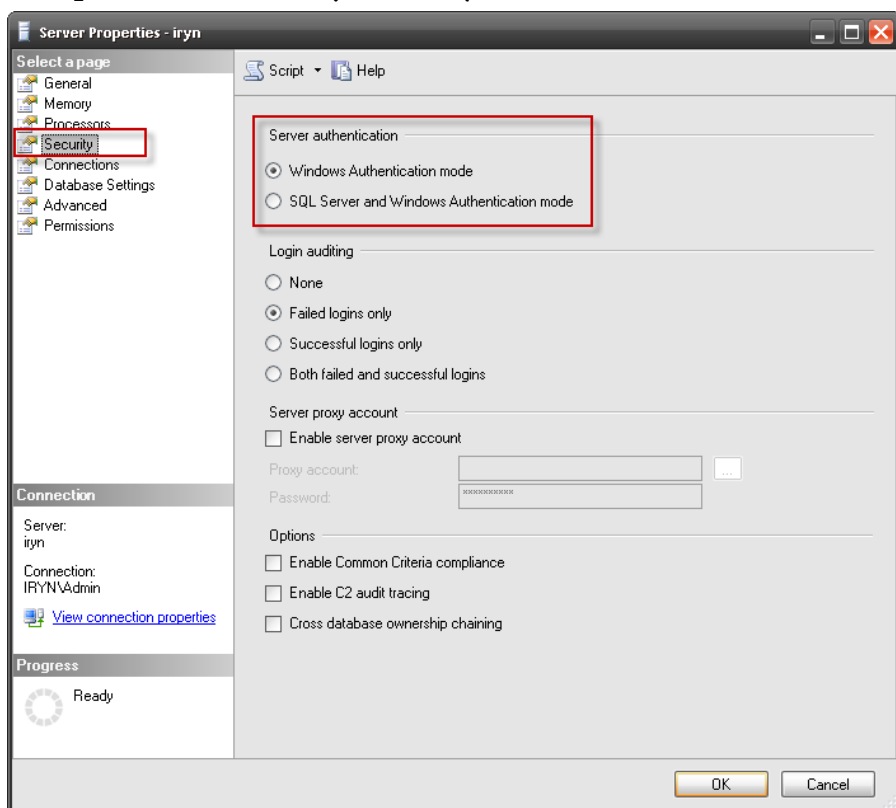
- пользователю не нужно запоминать много паролей, а достаточно одного – для входа на компьютер;
- проверка данных пользователя при входе на сервер происходит быстрее.

Но выбор в пользу смешанного режима может также перевесить:

- на предприятии не существует домена Windows, например, при использовании Unix систем;
- пользователи SQL Server не входят в домен, например, если они подключаются к серверу баз данных через Web-интерфейс.

Подытожим: режим защиты Windows наиболее безопасный, но смешанный режим защиты более гибкий и независимый от администратора сети.

Режим защиты данных можно изменить. Для этого в SSMS необходимо вызвать диалоговое окно Server Properties (Свойства сервера) отдельно взятого сервера и перейти на вкладку Security (Безопасность):



Кроме того, в системе SQL Server организована двухуровневая модель ограничения доступа к данным. В связи с этим порядок действий организации доступа к данным пользователя будет следующий:

1. Создать учетную запись или имя входа пользователя (login) на сервере, указав для него пароль и набор других параметров. Это позволит ему подключаться к самому серверу, но еще не дает доступа к базам данных. Данный этап является первым уровнем безопасности SQL Server (аутентификация пользователя).
2. К каждой необходимой базе данных добавить пользователя (user), который будет сопоставляться с именем входа. На основе прав пользователя баз данных (user), его имя входа (login) получает доступ к соответствующей базе данных. В связи с этим на одно имя входа может приходиться несколько пользователей (для разных баз данных), каждый из которых может иметь различные права доступа. Это второй уровень безопасности (авторизация пользователя).
3. Предоставить пользователю соответствующие права на базу данных.

Рассмотрим все по порядку.

2.2. Создание имен входа

Итак, для организации безопасности на сервере баз данных необходимо в первую очередь правильно спроектировать систему управления пользователями. В языке SQL нет набора стандартных команд, которые предназначены для создания имен входа и пользователей базы данных. В каждой СУБД для этого предназначен свой набор команд, но их суть во всех реализациях одинакова. В SQL Server 2008 для таких целей можно воспользоваться средствами SSMS или набором операторов T-SQL.

В T-SQL для создания имен входа (учетных записей, логинов) на сервере используют оператор CREATE LOGIN

```
CREATE LOGIN логин { WITH список_опций | FROM источник_
данных }

-- Список опций:
PASSWORD = { 'пароль' | <хешируемый_пароль' HASHED } [
MUST_CHANGE ]
[, SID = sid ] -- идентификатор
безопасности; GUID нового
- имени входа SQL Server
[, DEFAULT_DATABASE = БД_по_умолчанию ] -- база
данных, будет текущей
- при входе пользователя
[, DEFAULT_LANGUAGE = язык_по_умолчанию ]
[, CHECK_EXPIRATION = { ON | OFF } ] -- необходимо ли
устанавливать срок действия пароля
[, CHECK_POLICY = { ON | OFF } ] -- применять ли к логину
политику паролей
- Windows на компьютере, где выполняется SQL Server
[, CREDENTIAL = имя_учетных_данных ]

-- Источник данных:
- для Windows аутентификации
WINDOWS [ WITH [ DEFAULT_DATABASE = БД_по_умолчанию]
[, DEFAULT_LANGUAGE = язык_по_умолчанию] ]
| CERTIFICATE имя_сертификата
| ASYMMETRIC KEY имя_асимметричного_ключа
```

При создании логина сначала нужно указать его имя, которое может содержать от 1 до 128 символов (буквы, цифры и все символы, кроме символа обратного слэша (\)). В качестве имени входа нельзя использовать зарезервированные имена, а их значение не может быть NULL или являться пустой строкой ("). Следует отметить, что в SQL Server 2008 существует 4 типа имен входа:

- 1) имена входа SQL Server;
- 2) имена входа Windows, которые сопоставляются с учетной записью домена Windows. Формат таких имен выглядит [домен \ логин];
- 3) имена входа, которые сопоставлены с помощью сертификата;
- 4) имена входа, которые сопоставлены с помощью асимметричного ключа.

Для того чтобы создать имя входа SQL Server, необходимо выбрать конструкцию WITH. Для создания одного из трех последних типов подходит конструкция FROM оператора CREATE LOGIN.

Остановимся кратко на объяснении последних двух типов имен входа, а именно на именах входа, которые сопоставляются с помощью сертификата и асимметричного ключа. Начать, наверное, стоит с того, что SQL Server 2008 поддерживает иерархическую структуру ключей, которые позволяют шифровать данные. Сертификаты и асимметричные ключи являются одним из средств такого шифрования. Они должны содержаться в базе данных master и быть связаны с соответствующим пользователем.

Асимметричный ключ (asymmetric key) – это комбинация закрытого и открытого ключа, который ему соответствует. Для его создания используется оператор CREATE ASYMMETRIC KEY. Сертификаты (certificates) представляют собой подписанную цифровой подписью инструкцию, которая связывает значение открытого ключа с идентификатором пользователя (устройства или службы), который имеет соответствующий закрытый ключ. В SQL Server 2008 можно создавать собственные подписанные

сертификаты, которые соответствуют стандарту X.509 с помощью инструкции CREATE CERTIFICATE. Сертификаты являются надежным способом шифрования данных, хотя вместе с асимметричным ключом является ресурсоемких и поэтому медленным методом. Самый быстрый метод шифрования данных – это симметричный ключ. Более подробно о конфигурации шифрования в SQL Server можно прочитать в разделе "Иерархия средств шифрования" электронной документации SQL Server 2008.

Замечания по созданию имен входа SQL Server:

- если параметр HASHED не указан, тогда пароль хешируется перед сохранением в базу данных. Данный параметр не рекомендуется использовать при создании новых логинов. Он может быть полезен только в конкретных случаях, например, при переносе базы данных с одного сервера на другой;
- база данных по умолчанию – master;
- параметр MUST_CHANGE указывает на то, что при первом использовании нового имени входа, SQL Server будет осуществлять запрос пароля. Но здесь существует один нюанс: окно для изменения пароля появляется только в SQL Server Management Studio. Большинство других приложений при подключении такого пользователя вернут ошибку;
- параметр CREDENTIAL на сегодняшний день только связывает с новым именем входа объект Credential, который представляет собой пару "имя учетной записи Windows – пароль". Этот объект используется преимущественно в случае, когда пользователь хочет осуществить определенные действия

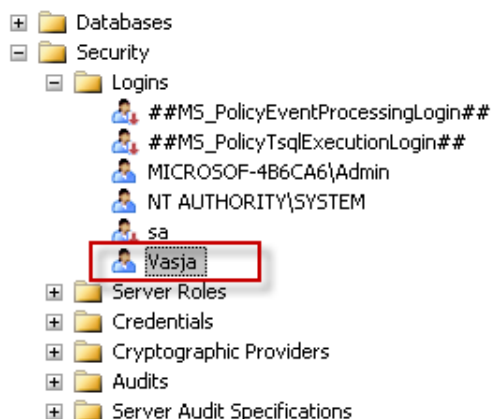
в ОС или на другом сервере SQL Server из сценария T-SQL. Объект Credential можно создать с помощью оператора CREATE CREDENTIAL;

- Параметр CHECK_POLICY будет работать только, если SQL Server работает под управлением Windows 2003 Server или более поздними версиями. Если он установлен, то на пароль для данного логина будут просто распространяться те требования, которые применяются к локальным учетным записям на данном компьютере.

Приведем пример создания логина SQL Server с паролем.

```
create login Vasja  
with password = '123';
```

Новый аккаунт будет размещен в директории Logins (логины, имена входа) папки Security (безопасность) сервера.



А теперь создадим имя входа Windows, которое будет связано с учетной записью Windows Pupkin из группы students, которая находится в домене stepdom:

```
create login [stepdom\students\Pupkin]
from windows;
```

Если необходимо зарегистрировать всех пользователей из группы students домена stepdom, кроме пользователя Petrenko, следует использовать оператор DENY (более подробно он будет рассмотрен ниже).

```
create login [stepdom\students] from windows;
-- регистрируем группу пользователей
deny connect sql to [stepdom\students\Petrenko];
-- исключаем из нее пользователя Petrenko
```

Список всех имен входа размещается в системном представлении sys.sql_logins. Кроме того, для просмотра списка всех имен входа на сервере можно воспользоваться системной хранимой процедурой безопасности sp_helplogins:

```
sp_helplogins [ [ @LoginNamePattern = ] 'ЛОГИН' ]
```

Например:

```
exec sp_helplogins 'sa';
```

Results

Messages

	LoginName	SID	DefDBName	DefLangName	AUser	ARemote
1	sa	0x01	master	us_english	yes	no

	LoginName	DBName	UserName	UserOrAlias
1	sa	D:\DB\PRESS.MDF	db_owner	MemberOf
2	sa	D:\DB\PRESS.MDF	dbo	User
3	sa	master	db_owner	MemberOf
4	sa	master	dbo	User
5	sa	model	db_owner	MemberOf
6	sa	model	dbo	User
7	sa	msdb	db_owner	MemberOf
8	sa	msdb	dbo	User
9	sa	tempdb	db_owner	MemberOf
10	sa	tempdb	dbo	User

Результатом работы данной системной хранимой процедуры будет 2 отчета:

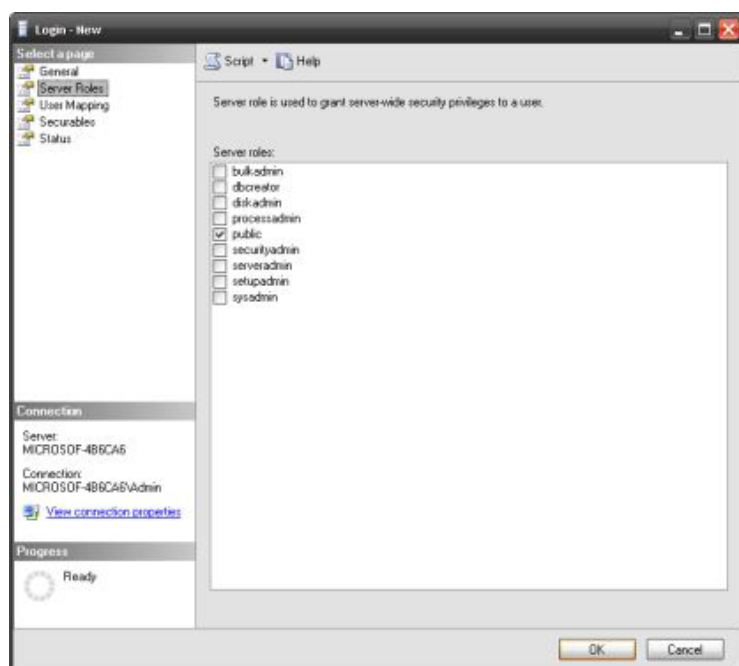
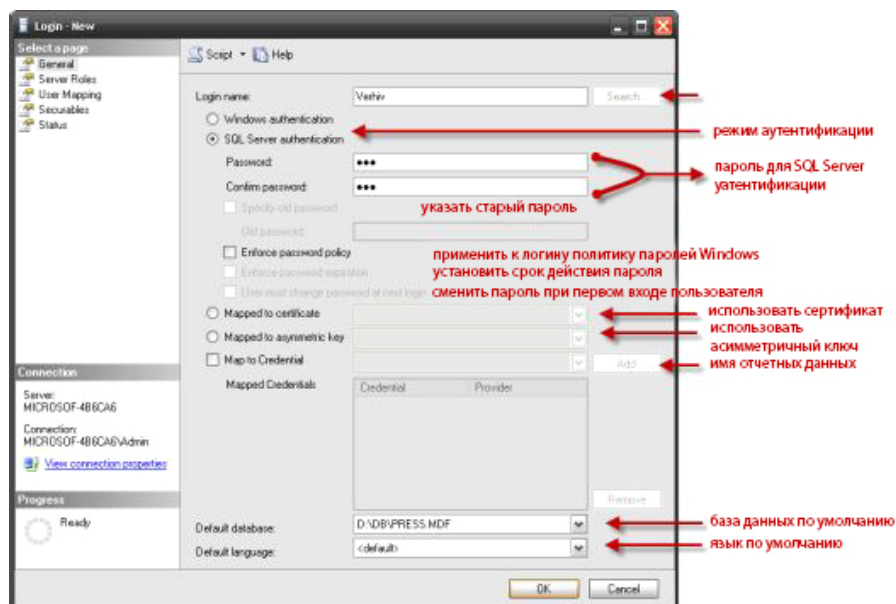
1. Первый отчет содержит данные об имени входа.
2. Второй отчет содержит обобщенные данные о пользователях, которые соответствуют данному имени входа и список ролей, членом которых он является.

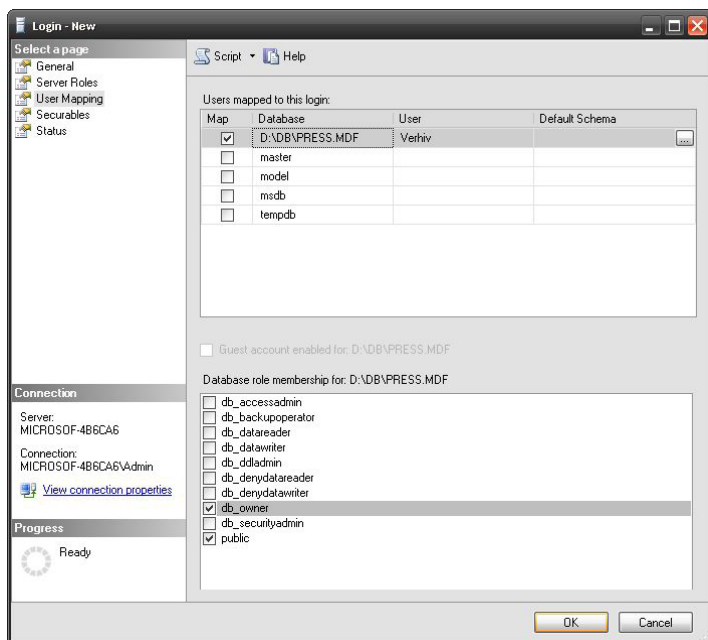
В SSMS папка Security-> Logins, которая содержит набор имен входа сервера баз данных. Чтобы добавить новый логин необходимо вызвать пункт контекстного меню New-> Login ... ветви Security или New Login ... дочернего узла Logins.



После этого появится диалоговое окно создания нового имени входа. На закладке General (Общие) необходимо указать общие свойства для нового имени входа. В первую очередь необходимо задать именно имя входа и пароль для него. Если вы создаете имя входа Windows, тогда можно воспользоваться кнопкой Search (Поиск) для выбора существующей учетной записи с Active Directory. В случае создания имени входа с сертификатом или асимметричным ключом следует выбрать существующий объект шифрования с помощью выпадающих списков.

Следующая закладка Server Roles (Серверные роли) предназначена для того, чтобы указать, каким фиксированным серверным ролям будет принадлежать имя входа. По умолчанию она принадлежит только роли PUBLIC. Тема ролей, включая и фиксированные серверные роли рассматривается в разделе "Роли" данного урока.





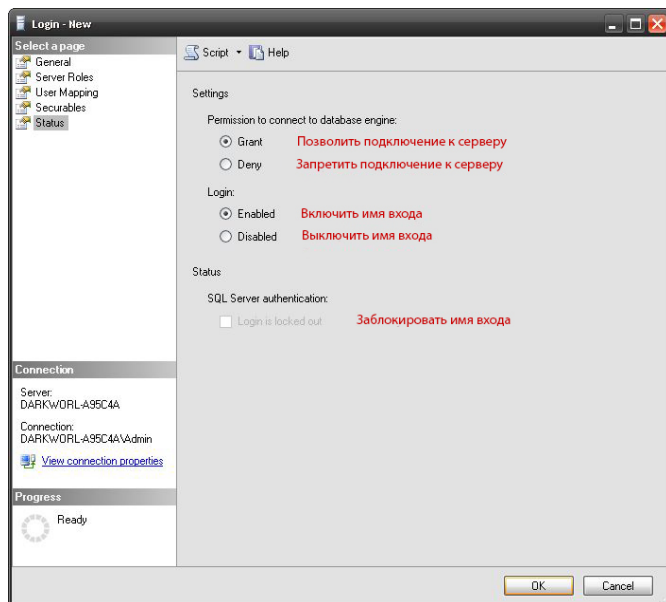
На закладке сопоставления пользователей (User Mapping) выберите базу данных Press, которую мы с вами создавали. После этого необходимо назначить права на эту базу данных для новосозданного имени входа. Например, предоставим ему полноценные права на базу данных Press, то есть назначим его роли db_owner. О роли уровня базы данных рассказывается в разделе "Роли".

На вкладке Securables (Защита) вы можете сразу установить права на:

- выбранный сервер и его объекты;
- конечные точки HTTP (Endpoints);
- другие имена входа (logins) и их объекты.

Например, вы можете сразу создать пользователя, который сможет менять пароли для других имен входа, которые закреплены за работниками его отдела.

Последняя вкладка Status (статус, состояние) позволяет настроить дополнительные параметры для нового имени входа.



Параметры здесь следующие:

1. **Permissions to connect to database engine (разрешение на подключение к ядру баз данных)** – значением по умолчанию для всех логинов является Grant, что указывает на разрешение подключения к SQL Server. Значение Deny устанавливается, как правило, только в случае раздачи прав именам входа группы Windows, но при этом одному или нескольким членам этой группы планируется запретить доступ к серверу.
2. **Login (Имя входа)**, который определяет состояние имени входа после создания: включен или нет.

Обычно все логины включены, но основания для отключения также существуют. Например, увольнение работника или перевода на его другую должность и т. Самым быстрым способом решить данную ситуацию будет простое отключение логина.

- 3. Status-> SQL Server authentication-> Login is locked out (Заблокировать логин).** Вы можете снять флажок блокировки имени входа, если оно заблокировано. Имя входа автоматически блокируется после использования всех попыток неверного ввода пароля для указанного логина, а также, если такое блокирование настроено на уровне операционной системы и для логина установлена опция CHECK_POLICY.

После нажатия кнопки ОК созданное вами имя входа будет отображаться в папке Security-> Logins, как и раньше.

В SQL Server 2008 существует набор встроенных логинов:

- NT AUTHORITY \ NetworkService – имя входа сетевой службы, которое используется для подключения к SQL Server Reporting Services. Он автоматически имеет права на базы данных master, msdb и на базы данных и службы, которые использует Reporting Services. Для SQL Server Agent данное имя входа использовать не рекомендуется.
- NT AUTHORITY \ LocalService – имя входа локальной службы, которое имеет ограниченный набор прав доступа к ресурсам (на уровне группы "Users" (Пользователи)). Этот логин обращается к сетевым ресурсам в форме нулевого сеанса без учетных данных. Имя локальной службы доступно только

в Microsoft Windows XP и Microsoft Windows Server 2003, а также не поддерживается SQL Server Agent.

- NT AUTHORITY \ System – локальная системная учетная запись операционной системы. Она появляется, если при инсталляции настроить работу службы SQL Server от имени локального системного аккаунта. Она является членом группы администраторов Windows на локальном компьютере и членом фиксированной серверной роли sysadmin. Имя NT AUTHORITY \ System существует только для обратной совместимости.
- sa (System Administrator) – имя входа, которое имеет полные и неотъемлемые права системного администратора SQL Server. Это имя входа удалить нельзя, его разрешается только переименовывать или выключать.
- существует также набор логинов, созданных на основе сертификата, имена которых ограничены двойным символом "хэш" (##). Такие имена входа используются только для внутреннего системного использования и удалять их не рекомендуется:
 - ## MS_SQLResourceSigningCertificate ##;
 - ## MS_SQLReplicationSigningCertificate ##;
 - ## MS_SQLAuthenticatorCertificate ##;
 - ## MS_AgentSigningCertificate ##;
 - ## MS_PolicyEventProcessingLogin ##;
 - ## MS_PolicySigningCertificate ##;
 - ## MS_PolicyTsqlExecutionLogin ##.

2.3. Модификация и удаление имен входа

Для изменения свойств имен входа используется оператор ALTER LOGIN, синтаксис которого представлен ниже:

```
ALTER LOGIN логин
{
    { ENABLE | DISABLE }          -- разрешить или запретить
    использование имени входа
    | WITH список_опций

    - добавить (или удалить) к имени входа учетные данные
    - поставщика ЭКМ (Extensible Key Management -
    расширенного управления ключами)
    | { ADD CREDENTIAL имя_учетных_данных | DROP CREDENTIAL
    имя_учетных_данных }
}

-- Список опций:
[ PASSWORD = { 'пароль' | 'хешированный_пароль' HASHED }
    [ OLD_PASSWORD = 'старый_пароль'
        | опции_пароля [ MUST_CHANGE | UNLOCK /*
    разблокировать логин */
    ]
]
[, DEFAULT_DATABASE = БД_по_умолчанию ]
[, DEFAULT_LANGUAGE = язык_по_умолчанию ]
[, NAME = логин ]                -- новое имя для логина,
которое необходимо переименовать
[, CHECK_POLICY = { ON | OFF } ] -- применять к логина
политику паролей
- Windows на компьютере, где выполняется SQL Server
[, CHECK_EXPIRATION = { ON | OFF } ] -- необходимо
устанавливать срок действия пароля
[, CREDENTIAL = имя_учетных_данных ]
[, NO CREDENTIAL] -- удаляет существующие сопоставления
логина и учетных данных сервера
```

Параметр DISABLE оператора ALTER LOGIN нельзя использовать для запрета доступа группы Windows. Для этого предназначен оператор DENY, который будет рассмотрен ниже.

Иногда параметр ENABLE путают с параметром UNLOCK. Чтобы не было путаницы, поясним подробнее. Параметр ENABLE позволяет использование имени входа, которое временно было запрещено администратором. А параметр UNLOCK разблокирует имя входа, доступ которого был запрещен сервером через использование всех попыток неверного ввода пароля для указанного логина. При этом разблокировать имя входа можно с новым паролем, а можно оставить ему старый, тогда как с помощью ENABLE изменить пароль нельзя, поскольку учетная запись была просто временно отключен.

Приведем несколько примеров:

```
-- изменим пароль, а именно отменим его
alter login Vasja
with password = '';

-- переименовать имя входа
alter login Vasja
with name = VPupkin;

-- разблокировать имя входа с новым паролем
alter login Vasja
with password='111' unlock;
```

Для удаления имени входа используется оператор DROP LOGIN:

```
-- синтаксис
DROP LOGIN логин
```



```
-- например, необходимо удалить имя входа Windows  
drop login [stepdom\students\Petrenko];
```

Стоит отметить, что для создания и управления именами входа можно использовать системные хранимые процедуры безопасности. Например, `sp_addlogin`, `sp_defaultdb`, `sp_droplogin`, `sp_grantlogin` и тому подобное. Но все эти хранимые процедуры считаются устаревшими и существуют в SQL Server 2008 только для обеспечения обратной совместимости. В следующей версии эти хранимые процедуры будут удалены.

3. Пользователи базы данных

Итак, имя входа для пользователя создали. Но этого, к сожалению, недостаточно для того, чтобы доступиться к данным в SQL Server. Пользователь должен иметь в необходимой базе данных еще и своего пользователя баз данных (database user). Такое разделение имен входа (login) и пользователей баз данных (user) в SQL Server обеспечивает большую гибкость. Теперь пользователь, имея один логин, может иметь несколько пользователей в различных базах данных и с различными правами доступа. Таким образом, пользователь после аутентификации на сервере проходит так называемую авторизацию пользователя. В ходе авторизации SQL Server проверяет, отвечает ли зарегистрированное имя входа (login) хотя бы одному из существующих пользователей базы данных (user), в которую планируется доступ, и какие права он на нее нет.

Создать пользователя базы данных можно с помощью оператора T-SQL CREATE USER:

```
CREATE USER имя_пользователя
[ { { FOR | FROM }
  { LOGIN имя_входа -- имя входа SQL Server,
    для которого создается пользователь базы данных
    | CERTIFICATE имя_сертификата -- сертификат, для
    которого создается пользователь БД
    | ASYMMETRIC KEY имя_асимметричного_ключа
  -- асимметричный ключ, для которого создается пользователь
  базы данных } }
```

```
| WITHOUT LOGIN          -- создает пользователя, который
будет - подключиться к базе данных как пользователь guest
]
[ WITH DEFAULT_SCHEMA = название_схемы ]-- схема по
умолчанию для пользователя БД, иначе dbo
```

Замечания по созданию пользователя:

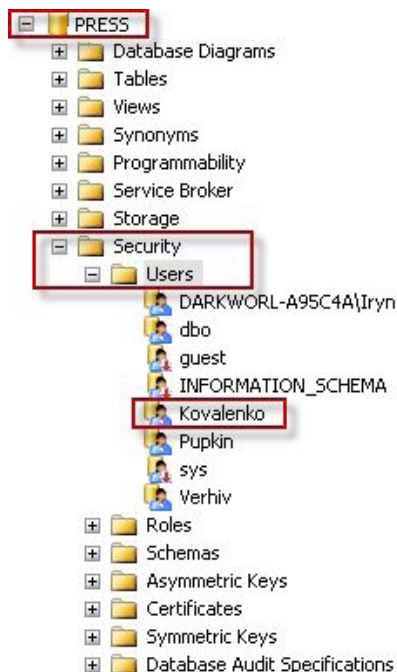
- если не указать имя входа, то есть не задать аргумент FOR LOGIN, тогда новый пользователь базы данных будет сопоставляться с именем входа SQL Server с аналогичным именем.
- значение DEFAULT_SCHEMA нельзя указывать для пользователей, которые сопоставляются с группами Windows, сертификатом или асимметричным ключом. Оно также игнорируется членами серверной роли sysadmin, поскольку для них схеме по умолчанию установлена схема dbo.

Например, создадим пользователя базы данных для имени входа Kovalenko.

```
create login Kovalenko
with password = 'qwerty';

use Press;
create user Kovalenko
for login Kovalenko;
-- или
- Create user Kovalenko;
Go
```

Новые пользователи базы данных добавляются в группу Users узла Security текущей базы данных. Итак, чтобы создать пользователей базы данных средствами SSMS, следует вызвать контекстное меню New User ветви Security или узла Users.



В случае доступа к базе данных имени входа, с которым не сопоставлен ни один пользователь базы данных, SQL Server предоставит доступ через пользователя guest, если у него есть соответствующие права. По умолчанию пользователь guest не имеет права доступа к базам данных, но вы можете ему их предоставить, включив его для необходимой базы данных:

```
-- выключить пользователя guest
grant connect to guest;
-- отменить доступ пользователя guest
revoke connect to guest;
```

Более подробно про операторы GRANT и REVOKE читайте в разделе "Управление правами доступа" данного урока.

Чтобы изменить свойства пользователя, необходимо воспользоваться оператором ALTER USER, а чтобы удалить пользователя базы данных – DROP USER.

```
-- изменить свойства пользователя баз данных
ALTER USER имя_пользователя
WITH [ NAME = новое_имя_пользователя ]
    [, DEFAULT_SCHEMA = название_схемы ]
    [, LOGIN = имя_входа ]

-- удалить пользователя баз данных
DROP USER имя_пользователя
```

Заметим, что для создания пользователей базы данных используется также системная хранимая процедура sp_grantdbaccess, но она считается устаревшей и в следующей версии SQL Server будет удалена, поэтому использовать ее не стоит. Данные обо всех пользователях базы данных хранятся в системном представлении sys.server_principals.

На уровне базы данных выделяют также понятие "осиротевших" пользователей (orphaned users), которые представляют собой пользователей базы данных, с которыми не сопоставляется ни одно имя входа. В эту категорию попадают пользователи при удалении связанных с ними имен входа. Для получения информации о таких пользователях используется системная хранимая процедура sp_change_users_login.

```
-- синтаксис хранимой процедуры sp_change_users_login
sp_change_users_login [ @Action = ] 'действие'
                    [ , [ @UserNamePattern = ] 'пользователь' ]
                    [ , [ @LoginName = ] 'имя_входа' ]
                    [ , [ @Password = ] 'пароль' ]
```

В качестве действий можно указывать одно из следующих значений:

- `Auto_Fix` – связывает запись пользователя в системном представлении `sys.database_principals` текущей базы данных с именем входа SQL Server, который имеет такое же имя. С точки зрения безопасности, старайтесь избегать инструкции `Auto_Fix`, поскольку при отсутствии необходимого имени входа, оно создается.

При использовании `Auto_Fix` необходимо соблюдать следующие правила:

- если имени входа еще не существует, тогда необходимо указать аргументы `user` и `password`;
- если имя входа уже присутствует, тогда следует указать аргумент `user` без `password`;
- аргумент `login` должен иметь значение `NULL`.
- `Report` – выводит список "осиротевших" пользователей и их идентификаторы безопасности (SID) в текущей базе данных. При этом аргументы `user`, `login` и `password` должны иметь значение `NULL` или отсутствовать.
- `Update_One` – связывает указанный аргумент `UserNamePattern` в текущей базе данных с существующим аргументом `LoginName`. Аргумент `password` должен отсутствовать или равен `NULL`.

К примеру:

```
exec sp_change_users_login @action = 'REPORT'
```

Результат:

Results		Messages
	UserName	UserSID
1	Pupkin	0xB0B3D59AF6FDF44CBD14F7005CCEEB26

При создании новой базы данных, в ней автоматически создаются 4 пользователя:

- **dbo** (database owner) – владелец базы данных, которым автоматически становится тот логин, от имени которого была создана база данных. Он входит в фиксированную роль базы данных **db_owner** и имеет все права на базу данных.
- **guest** (гость) – пользователь, который дает права всем логинам, которым не соответствует ни один пользователь базы данных.
- **INFORMATION_SCHEMA** – владелец схемы **INFORMATION_SCHEMA**, в которой хранятся представления с системной информацией для базы данных. Данному пользователю не соответствует ни одно имя входа.
- **sys** – владелец схемы **SYS**, в которую входят системные объекты базы данных. Данному пользователю не соответствует ни одно имя входа.

4. Роли

4.1. Понятие ролей. Фиксированные роли

А теперь время рассмотреть еще одних участников системы безопасности сервера – роли, которые позволяют определить уровень доступа (права) к объектам сервера баз данных. По сути, роль – это группа пользователей, которые имеют одинаковые права. Например, если у нас существует группа пользователей, которым нужен доступ только для чтения, то мы можем создать роль `READER`, присвоить ей необходимые права, а затем назначать эту роль конкретному пользователю. То есть, если роли предоставляются какие-либо права, то эти же права имеет каждый пользователь, который входит в данную роль.

Кроме того, в `SQL Server 2008` пользователь может входить в состав нескольких ролей. Это позволяет группировать права доступа в определенные роли, а затем использовать их комбинации, создавая тем самым наборы прав, которые наиболее подходят конкретному пользователю. В связи с этим роли сервера часто сравнивают с группами пользователей `Windows`.

Отметим, в каждой базе данных, включая системные базы данных, такие как `master`, `model` и другие, существует роль `PUBLIC`. Это специальная роль, которая предназначена для предоставления прав сразу всем пользователям базы данных. Она имеет настроенные по умолчанию права для пользователей базы данных и ни ее члены, ни она сама

не может быть уничтожена. Ее членом автоматически становится каждый пользователь базы данных.

Все остальные роли можно разделить на следующие виды:

1. Роли уровня сервера (почти все имена заканчиваются на xxxadmin) – это роли, которые действуют в рамках всего сервера и не принадлежат одной базе данных. Они предназначены для сопровождения системы, включая управление учетными записями и работу со связанными серверами. К встроенным (фиксированным) серверным ролям относятся:
 - sysadmin – пользователи данной роли могут выполнять любые действия с SQL Server, поскольку эта роль объединяет в себе права всех других серверных ролей;
 - serveradmin – пользователи данной роли могут выполнять действия по конфигурированию сервера: изменять параметры сервера, завершать его работу и тому подобное;
 - setupadmin – пользователи роли могут управлять (добавлять, настраивать и удалять) связанными серверами, а также выполнять ряд системных хранимых процедур. Им разрешается также объявлять процедуры запуска;
 - securityadmin – позволяет управлять именами входа сервера, изменять параметры настроек безопасности, включая удаленные серверы и права на создание базы данных, а также предоставляет доступ к журналу ошибок;

- processadmin – позволяет управлять процессами, которые выполняются в экземпляре SQL Server;
- diskadmin – пользователи данной роли могут управлять дисковыми файлами (присоединять и отсоединять базы данных, указывать, какие файлы входят в состав файловых групп и т.п.);
- bulkadmin – пользователи этой роли имеют право на выполнение оператора BULK INSERT, который используется для массовой вставки данных. При этом данная роль не предоставляет права доступа к таблицам, в которых будет применен данный оператор;
- dbcreator – позволяет создавать и управлять базами данных, включая восстановление из резервной копии.

Для того, чтобы добавить нового пользователя или группу в серверную роль используется системная хранимой процедура `sp_addsrvrolemember`, а для исключения одного из членов – `sp_dropsrvrolemember`.

```
-- добавить в роли
sp_addsrvrolemember [ @loginame= ] 'логин', [ @rolename =
] 'серверная_роль'

-- исключить из роли
sp_dropsrvrolemember [ @loginame = ] 'логин', [ @rolename
= ] 'серверная_роль'
```

Например:

```
-- добавляем к роли sysadmin пользователя Verhiv
exec sp_addsrvrolemember @loginame = 'Verhiv', @rolename =
'sysadmin'

-- отключить пользователя Verhiv с серверной роли
```

```
diskadmin
exec sp_dropserverolemember 'Verhiv', 'diskadmin'
```

Сведения о членах серверной роли можно получить из системного представления sys.server_role_member.

2. Роли уровня базы данных (их имена имеют префикс db_xxx) существуют на уровне базы данных и предназначены для определения прав по работе с конкретной базой данных. К встроенным ролям уровня базы данных относятся:

- db_owner – данная роль назначается владельцам базы данных, которые имеют на нее полные права;
- db_accessadmin – пользователи данной роли имеют право раздавать и отнимать права доступа к базе данных;
- db_backupoperator – набор прав для резервного копирования базы данных;
- db_datareader – данная роль объединяет все права по считыванию данных из таблиц, представлений и функций;
- db_datawriter – пользователи данной роли имеют права на добавление, обновление и удаление данных в текущей базе данных;
- db_ddladmin – пользователи данной роли имеют права на управление данным в базе данных (INSERT, DELETE, UPDATE);
- db_denydatareader – запрещает всем пользователям считывать данные базы данных (SELECT);
- db_denydatawriter – запрещает всем пользователям управлять данными таблиц и представлений текущей базы данных (INSERT, DELETE, UPDATE).

- db_securityadmin – данная роль объединяет в себе все права по администрированию системы защиты базы данных, то есть позволяет управлять членами ролей и их правами.

В SQL Server существует также возможность создавать собственные роли уровня базы данных с помощью оператора CREATE ROLE. Более подробно об этой возможности читайте в п.4.2. "Пользовательские роли" текущего раздела.

Чтобы добавить пользователя, имя входа или иную роль в качестве текущей базы данных, необходимо вызвать системную зберигаему процедуру sp_addrolemember:

```
sp_addrolemember [ @rolename = ] 'роль', [ @membername = ]
'учетная_запись'
```

Например, необходимо в качестве уровня базы данных добавить имя входа Windows.

```
exec sp_addrolemember 'db_owner', [Stepdom\Students\
Pupkin];
```

При этом в роли уровня базы данных нельзя добавлять другие фиксированные роли уровня базы данных, фиксированные серверные роли или пользователя dbo.

Удалить пользователя, имя входа или иную роль с роли уровня базы данных помогает серверная хранимая процедура sp_droprolemember.

```
sp_droprolemember [ @rolename = ] 'роль', [ @membername =
] 'учетная_запись'
```

3. Роли уровня приложения позволяют администратору базы данных ограничивать доступ пользователей к данным через прикладное приложение. Эта роль не содержит в себе пользователей, поэтому перед

использованием ее необходимо активизировать в текущем соединении, введя верный пароль.

Процесс работы роли уровня приложения происходит следующим образом:

- 1) На сервере баз данных создается роль уровня приложения (CREATE APPLICATION ROLE) и ей назначается определенный набор прав.
- 2) Сначала пользователь регистрируется в самом прикладном приложении предусмотренным для этого способом, например, с помощью диалогового окна.
- 3) Пользователь проверяется на существование и ему назначаются установленные права.
- 4) Роль уровня приложения активизируется (системная хранимая процедура `sp_setapprole`).
- 5) Проверяется роль уровня приложения и соединение переключается в контекст этой роли. Стоит отметить, что при активизации роли уровня приложения все права ролей пользователей на сервере игнорируются и они получают только права, которые действуют на уровне прикладного приложения. В связи с этим, если пользователю необходимо снова вернуться к заданным для него параметрам безопасности, он должен закрыть соединение с сервером и зарегистрироваться на сервере снова.

Итак, создать роль уровня приложения можно с помощью оператора CREATE APPLICATION ROLE:

```
CREATE APPLICATION ROLE название_роли
WITH PASSWORD = 'пароль'
[, DEFAULT_SCHEMA = название_схемы ]
```

Аргумент DEFAULT_SCHEMA указывает на схему по умолчанию, то есть на первую схему, в которой сервер будет искать объекты для этой роли. Если она не задана, то схемой по умолчанию становится схема dbo. Кстати, указанная схема в базе данных может временно отсутствовать.

Что касается паролей ролей уровня приложения, то они должны храниться в зашифрованном виде.

После создания роли уровня приложения, чтобы ею воспользоваться ее необходимо активизировать с помощью хранимой процедуры sp_setapprole:

```
sp_setapprole [ @rolename = ] 'название_роли'
[, @password = ] { encrypt N'пароль_роли' } |
'password'
[, [ @encrypt = ] { 'none' | 'odbc' } ]
[, [ @fCreateCookie = ] true | false ] -- нужно
создавать cookie файл
[, [ @cookie = ] @cookie OUTPUT ] -- выходной
параметр
-- (тип
varbinary(8000))
```

Аргумент encrypt позволяет указать тип шифрования пароля, но только для соединений, которые не используют приложение SqlConnection:

- none – кодирование пароля не используется, то есть пароль передается SQL Server в виде обычного текста (по умолчанию);
- odbc – закодировать с помощью функции ODBC encrypt перед отправкой компонента SQL Server

Database Engine. Но для защиты паролей, которые передаются по сети, ее использовать не следует, поскольку она их не шифрует. Следует отметить, что данный аргумент может быть указан только для клиентов ODBC или поставщика данных OLE DB для SQL Server.

При использовании функции encrypt пароль должен быть преобразован в строку Unicode с помощью модификатора N. Если учетные данные необходимо хранить в клиентском приложении, тогда их лучше зашифровывать с помощью функций шифрования API.

После включения роли уровня приложения она остается активной до тех пор, пока пользователь не отключится от сервера или не выполнит хранимую процедуру sp_unsetapprole.

Приведем небольшой пример создания роли приложения и ее активации:

```
-- создаем роль уровня приложения
create application role QRole
with password = 'AsDeF00MbXX';

-- активизируем ее
exec sp_setapprole 'QRole', 'AsDeF00MbXX';
```

Изменить свойства роли приложения можно с помощью оператора ALTER APPLICATION ROLE.

```
-- синтаксис
ALTER APPLICATION ROLE имя_роли
WITH [ NAME = новое_имя_роли ]
[ , PASSWORD = 'пароль' ]
[ , DEFAULT_SCHEMA = название_схемы ]
```

```
-- например, добавим к роли приложения Role схему по
умолчанию
alter application role QRole
with default_schema = sale;
```

Для удаления роли уровня приложения используют оператор DROP APPLICATION ROLE.

```
DROP APPLICATION ROLE имя_роли
```

Устаревшим аналогом оператора CREATE APPLICATION ROLE является системная хранимая процедура sp_addapprole. Напомним, что в SQL Server 2008 она существует только для совместимости с предыдущими версиями.

Роли уровня приложения существуют на уровне базы данных. То есть, если обратиться к другой базе данных, когда прикладное приложение находится в контексте безопасности роли приложения, тогда доступ к ней будет осуществляться с учетом прав учетной записи guest в этой базе данных. Если учетная запись guest неактивна для этой базы данных, доступ к ней будет запрещен.

Роли приложения действуют также во время сеансов. Итак, если ваш прикладное приложение работает со многими сеансами, которые используют одну роль, в таком случае каждый сеанс должен ее сначала активизировать.

Эта роль очень полезна в случаях, когда на уровне сервера необходимо ограничить доступ пользователей, передав полномочия в распределении прав определенному прикладному приложению. Например, теперь клиентское прикладное приложение может для одних подключений использовать контекст безопасности пользователя, а для других контекст безопасности роли приложения.

При использовании роли уровня приложения могут быть полезными следующие функции:

- `USER_NAME ([id])` – возвращает имя текущего пользователя или имя пользователя с указанным идентификатором;
- `SYSTEM_NAME` – возвращает имя текущего имени входа.

4.2. Пользовательские роли

Как уже было сказано ранее, SQL Server поддерживает возможность создания собственных ролей уровня базы данных, которые будут группировать пользователей баз данных с одинаковыми правами доступа к объектам. Это позволит классифицировать пользователей по категориям необходимости доступа.

Синтаксис оператора **CREATE ROLE**, который предназначен для создания пользовательской роли базы данных, имеет следующий вид:

```
CREATE ROLE имя_роли [ AUTHORIZATION собственник ]
```

По умолчанию собственником роли является собственник текущей базы данных, но его можно изменить на другого пользователя базы данных или сделать им другую роль. На практике, данный аргумент при создании роли почти не используется.

Устаревшим аналогом данного оператора является системная хранимая процедура **sp_addrole**.

После создания в роль необходимо добавить пользователей с помощью системной хранимой процедуры

sp_addrolemember, а удалить пользователя из роли – **sp_droprolemember**.

Например, создадим роль базы данных **Management** и добавим к ней пользователя **Verhiv**.

```
create role Management; -- создаем роль базы данных
exec sp_addrolemember 'Management', 'Verhiv'; -- добавляем
в него пользователя
```

Для изменения свойств роли следует воспользоваться оператором **ALTER ROLE**. Но спектр ее возможностей распространяется только на изменение названия роли.

```
ALTER ROLE название_роли
WITH NAME = новое_имя_роли
```

Для удаления пользовательской роли базы данных используется оператор **DROP ROLE**:

```
DROP ROLE название_роли
```

Создавать и управлять пользовательскими ролями можно и средствами SSMS. Все пользовательские роли после создания размещаются в ветке **Security-> Roles** соответствующей базы данных. Следовательно, и создавать их можно с помощью контекстного меню **New Role** узла **Roles** или родительского узла **Security**.

Добавить новую роль



```
-- синтаксис системной хранимой процедуры sp_helprole
sp_helprole [ [ @rolename = ] 'имя роли' ]

-- примеры:
- пересмотрим все роли текущей базы данных
exec sp_helprole;
-- информация о членах роли "Management"
exec sp_helprole 'Management';
```

Результат:

The screenshot shows the 'Roles' folder in SQL Server Enterprise Manager. The 'Roles' table is expanded, showing the following data:

RoleName	RoleId	IsAppRole
public	0	0
Management	9	0
db_owner	16384	0
db_accessadmin	16385	0
db_securityadmin	16386	0
db_dtladmin	16387	0
db_backupoperator	16389	0
db_datareader	16390	0
db_datawriter	16391	0
db_denydatareader	16392	0
db_denydatawriter	16393	0

A blue arrow points to the 'Management' role. Below the table, the command 'exec sp_helprole' is entered in the command window.

```
select * from sys.database_principals
```

Результат:

	name	principal_id	type	type_desc	default_schema_name	create_date	modify_date	owning_principal_id	sid	is_fixed_role
1	public	0	R	DATABASE_ROLE	NULL	2003-04-08	2005-10-14	1	0x	0
2	dbo	1	S	SQL_USER	dbo	2003-04-08	2003-04-08	NULL	0x	0
3	guest	2	S	SQL_USER	guest	2003-04-08	2003-04-08	NULL	0x	0
4	INFORMATION_SCHEMA	3	S	SQL_USER	NULL	2005-10-14	2005-10-14	NULL	0x	0
5	sys	4	S	SQL_USER	NULL	2005-10-14	2005-10-14	NULL	0x	0
6	NT AUTHORITY\SYSTEM	5	U	WINDOWS_USER	dbo	2003-04-08	2003-04-08	NULL	0x	0
7	Public	6	S	SQL_USER	dbo	2007-11-03	2007-11-03	NULL	0x	0
8	Vendor	7	S	SQL_USER	dbo	2010-09-27	2010-09-27	NULL	0x	0
9	Kovalevka	8	S	SQL_USER	dbo	2010-10-04	2010-10-04	NULL	0x	0
10	Management	9	R	DATABASE_ROLE	NULL	2010-10-10	2010-10-10	1	0x	0
11	db_owner	10384	R	DATABASE_ROLE	NULL	2010-04-08	2005-10-14	1	0x	1
12	db_accessadmin	16385	R	DATABASE_ROLE	NULL	2003-04-08	2005-10-14	1	0x	1
13	db_securityadmin	16386	R	DATABASE_ROLE	NULL	2003-04-08	2005-10-14	1	0x	1

Расшифровываем коротко структуру данного системного представления:


Поле	Описание
name	Участник базы данных (роль или пользователь).
principal_id	Идентификатор участника базы данных.
type	Тип участника: S – пользователь SQL; U – пользователь Windows; G – группа Windows; A – роль уровня приложения; R – роль уровня базы данных; C – пользователь, сопоставлен с сертификатом; K – пользователь, сопоставлен с асимметричным ключом.
type_desc	Описание типа участника.
default_schema_name	Схема по умолчанию. Равна NULL для участников, которые не принадлежат типа S, U или A.
create_date	Дата создания участника.
modify_date	Время последней модификации участника.
owning_principal_id	Идентификатор собственника участника. Собственником всех участников, кроме ролей базы данных, является dbo.
sid	Идентификатор защиты (CID), если участник принадлежит типа S, U или G, иначе NULL.

Поле	Описание
is_fixed_role	Флаг, который указывает на встроенную (фиксированную) роль.

Для просмотра членов ролей базы данных, используется системное представление **sys.database_role_members** и хранимая процедура **sp_helpuser**. Но **sp_helpuser** не возвращает информацию о защищенных объектах SQL Server 2008, поэтому для этих целей лучше воспользоваться системным представлением **sys.database_principals**.

```
exec sp_helpuser;
```

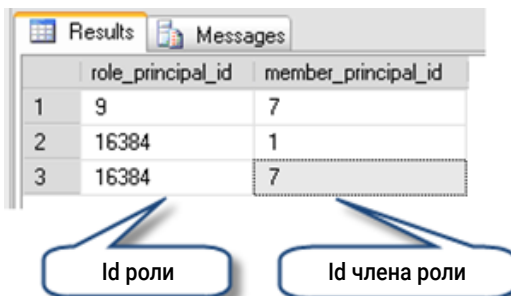
Результат:



	UserName	RoleName	LoginName	DefDBName	DefSchemaName	UserID	SID
1	DARKWORLD\SSCAVlyn	public	NULL	NULL	dbo	5	0x...
2	dbo	db_owner	sa	master	dbo	1	0x01
3	guest	public	NULL	NULL	guest	2	0x00
4	INFORMATION_SCHEMA	public	NULL	NULL	NULL	3	NU...
5	Kovalenko	public	Kovalenko	master	dbo	8	0x...
6	Pupkin	public	NULL	NULL	dbo	6	0x...
7	sys	public	NULL	NULL	NULL	4	NU...
8	Vetiviv	Management	Vetiviv	D:\DB\PRESS.MDF	dbo	7	0x...
9	Vetiviv	db_owner	Vetiviv	D:\DB\PRESS.MDF	dbo	7	0x...

```
select * from sys.database_role_members
```

Результат:



	role_principal_id	member_principal_id
1	9	7
2	16384	1
3	16384	7

Как видно из результата, к роли Management (role_principal_id = 9) текущей базы данных Press, входит один пользователь с именем Verhiv (member_principal_id = 9). Роль db_owner (role_principal_id = 16384) этой базы данных включает в себя двух пользователей: Verhiv и dbo (member_principal_id = 1).

5. Управление правами доступа

5.1. Основные понятия

После создания роли или конкретного пользователя и установки для них прав доступа к базе данных, им необходимо предоставить права (привилегии) на объекты самой базы данных. Это необходимо для организации дополнительного уровня безопасности данных. Например, на предприятии работникам отдела снабжения необходимо предоставить только доступ к данным, которые касаются клиентов, поставки и товара и запретить просматривать информацию о работниках предприятия, включая данные о зарплате 😊. Осуществляется такое управление правами с помощью операторов **GRANT**, **DENY** и **REVOKE**.

Права – это разрешение пользователю осуществить определенную операцию над объектом базы данных. Существуют ряд видов объектов и операций, на которые можно устанавливать права для пользователей и ролей.

Права доступа, которые получает пользователь, можно разделить на **следующие типы**:

1. **Неявные права** – это набор прав, которые определяются принадлежностью к определенной фиксированной роли (сервера, базы данных или роли уровня приложения) пользователя;

2. Объектные права – это права по управлению объектами базы данных;
3. Командные права – права на выполнение операций над базой данных.

5.2. Объектные права доступа

В SQL Server, начиная с версии SQL Server 2005, объектов в базе данных, на которые можно назначить права, стало больше. Теперь, кроме раздачи прав на пользование таблицами, представлениями, функциями и хранимыми процедурами, можно назначать права на управление такими объектами, как роли, пользователи базы данных, полнотекстовые каталоги и т. Таким образом, к защищенным объектам относятся все ресурсы, доступ к которым регулируется системой авторизации компонента SQL Server Database Engine. Кроме того, ряд защищенных объектов могут храниться внутри других, создавая так называемые иерархии областей, для которых можно устанавливать права доступа.

Таких **областей** в SQL Server 2008 **существует три**:

1. **Сервер**, который содержит в себе такие защищаемые объектов, как:
 - имя входа (login), то есть можно одному пользователю SQL Server разрешить доступ к объекту другого пользователя;
 - база данных;
 - конечные точки (endpoint) HTTP.

2. База данных, которая включает следующие защищаемые объекты:

- Пользователь (user);
- Роль;
- Роль уровня приложения;
- Сборка (assembly) .NET;
- Тип сообщений, который позволяет сдать разрешение на защищаемый объект компонента Service Broker;
- Маршрут;
- Служба;
- Привязка удаленной службы;
- Полнотекстовый каталог;
- Сертификат;
- Симметричный и асимметричный ключ;
- Контракт;
- Схема.

3. Схема, которая содержит такие защищаемые объекты:

- Пользовательский тип данных (домен);
- Коллекция схем XML;
- Объект:
 - таблица;
 - представление;
 - функция;
 - хранимая процедура;
 - ограничения;
 - очередь;
 - синоним;
 - статистика;
 - статистическое вычисление.

Это нововведение позволяет существенно облегчить работу администратору баз данных, поскольку предоставив пользователю права на область, например, на управление схемой, вы предоставляете ему сразу права на пользование всеми объектами этой схемы.

Права на объекты базы раздаются с помощью оператора **GRANT**. В нем очень много опций и полное знакомство с данным оператором выходит за пределы нашего урока. Мы же попробуем рассмотреть лишь основные принципы его использования. Обобщенный синтаксис оператора **GRANT** выглядит следующим образом:

```
GRANT { ALL [ PRIVILEGES ] }
      | название_привилегии [ ( название_поля [ ,...n ]
) ] [ ,...n ]
[ ON [ класс_защищаемого_объекта :: ] защищаемый_объект ]
TO имя_учетной_записи [ ,...n ]
[ WITH GRANT OPTION ]
[ AS имя_учетной_записи]
```

Ключевое слово **ALL** указывает на то, что пользователю необходимо предоставить все права на указанный объект. Но в SQL Server 2008 оно не предоставляет все права. Более того, ключевое слово **ALL** является устаревшим и существует только для совместимости с предыдущими версиями. В следующей версии данный параметр планируется исключить, поэтому использовать его не стоит.

Ключевое слово **PRIVILEGES** также является устаревшим и было введено в предыдущих версиях только для соответствия стандарту SQL-92.

Если ключевое слово **ALL** не используется, тогда указывается список прав доступа. После ключевого слова **ON** рассказываем на какой именно объект раздаются права и кому (**TO**). Получателем прав может быть пользователь базы данных, имя входа (включая имя входа или группу Windows) или роль.

Опция **WITH GRANT OPTION** указывает на то, что пользователи, которым предоставляются права на объект (ы) базы данных, могут предоставлять их другим пользователям.

Если учетная запись принадлежит нескольким ролям, тогда необходимо при раздаче прав указать опцию **AS**. После этого ключевого слова задается список ролей, которые конкретизируют пользователя: пользователю с какой именно роли предоставить те или иные права.

Как видно из синтаксиса, в SQL Server можно устанавливать права на уровне отдельных полей. Но на практике лучше не пользоваться такими правами из-за потери производительности и усложнение системы прав. Для таких целей, то есть запрета доступа пользователя или роли в определенных полях таблицы или иного объекта, лучше использовать представления или зберируемую процедуру, которые будут показывать информацию.

Например, разберем права, чаще всего раздаются, то есть права на пользование объектами базы данных. Итак, в SQL Server можно предоставлять следующие **объектные права**:

1. **SELECT, INSERT, UPDATE, DELETE** на таблицы, представления, табличные функции и на поля перечисленных объектов, а также на синонимы.

2. Право на запуск хранимых процедур и скалярных функций – **EXECUTE**.
3. Право **REFERENCES** двоякое, поскольку его можно предоставлять таблицам, представлениям, функциям, которые возвращают табличное значение, и очередям компонента Service Broker. Таким образом:
 - право **REFERENCES** для таблицы необходимо для создания ограничения на внешний ключ (**FOREIGN KEY**), который ссilaется на эту таблицу;
 - право **REFERENCES** для представления или табличной функции необходимо для сворення этих объектов с параметром **WITH SCHEMABINDING**.
4. **ALTER** – возможность изменять свойства объекта (по умолчанию это право имеет только его владелец). На уровне базы данных можно настраивать права **ALTER ANY объект_сервера_або_БД**, которые предоставляют возможность вносить изменения в произвольные объекты сервера или базы данных. Например, **ALTER ANY LOGIN** дает право создавать, изменять или удалять произвольное имя входа в экземпляре, а **ALTER ANY TABLE** позволяет управлять таблицами базы данных.

Следует также помнить, что если вы установите право **ALTER** на схему, это позволит управлять всеми объектами, входящими в эту схему.
5. **CONTROL** – предоставляет полные права на объект, включая информацию в нем. Кроме того, пользователю, который имеет право **CONTROL**,

разрешается раздавать права на установленный объект другим. Право CONTROL на определенную область неявно включает право CONTROL на все объекты, которые находятся в ее пределах. Например, привилегия CONTROL на базу данных неявно предоставляет все права на базу данных, все права на ее сборки (assembly), на ее схемы и объекты, которые находятся в пределах этих схем.

6. **TAKE OWNERSHIP** – право на передачу прав собственности на объект. Такое право можно назначить для любых объектов.
7. **VIEW DEFINITION** – право на просмотр метаданных таблиц, представлений, процедур или функций.
8. **VIEW CHANGE TRACKING** – право на отслеживание изменений в таблице (или схеме), например, будет предоставлена информация о том, какие поля были изменены. Полный список операций, которые отслеживаются следующий:
 - DROP TABLE;
 - ALTER TABLE DROP CONSTRAINT;
 - ALTER TABLE DROP COLUMN;
 - ALTER TABLE ADD COLUMN;
 - ALTER TABLE ALTER COLUMN;
 - ALTER TABLE SWITCH;
 - DROP INDEX или ALTER INDEX DISABLE;
 - TRUNCATE TABLE.
9. **RECEIVE** – право на работу с очередями компонента Service Broker.

Чтобы лучше понять работу оператора GRANT, рассмотрим несколько **примеров**:

1. Предоставить права на выборку данных из таблицы book.Authors пользователю Verhiv:

```
grant select
on book.Authors
to Verhiv;
```

2. Если необходимо предоставить права сразу нескольким пользователям:

```
grant select
on OBJECT::book.Authors
to Verhiv, Kovalenko;
```

3. Предоставить роли Management права на просмотр и изменение данных таблицы sale.Sales:

```
grant select, update
on OBJECT::sale.Sales
to Management;
```

4. Если необходимо выдать права на просмотр и выборку данных таблицы sale.Shops всем пользователям текущей базы данных, тогда можно использовать роль PUBLIC, которая эквивалентна перечню всех пользователей.

```
grant select, insert
on OBJECT::sale.Shops
to public;
```

5. Для ограничения прав на просмотр таблицы `sale.Sales` лишь несколькими полями, необходимо написать следующий запрос.

```
grant select on sale.Sales to public;  
grant update  
on sale.Sales (Price, Quantity)  
to Kovalenko  
with grant option;
```

В результате все пользователи могут только просматривать таблицу `sale.Sales`, а пользователь `Kovalenko` имеет еще право изменять в ней поля `Price` и `Quantity`. Кроме того, он имеет право передавать эти привилегии другим пользователям.

6. Предоставим право на запуск хранимой процедуры `sp_BestAuthors` роли `Secretary`:

```
grant execute  
on OBJECT::sp_BestAuthors  
to Secretary;
```

Оператор **DENY** позволяет установить запрет на использование объектов базы данных. Особенностью его использования является то, что он является высшим по приоритету, чем оператор **GRANT**, поэтому позволяет отменить права пользователя, выданные ранее. Но запрет (**DENY**) на уровне таблицы имеет меньший приоритет, чем разрешение уровня поля, выданный оператором **GRANT** (!). Эта несовместимость прав в SQL Server 2008 существует только для обеспечения обратной совместимости и в следующей версии она будет устранена.

Полный синтаксис оператора DENY также довольно сложный и несколько напоминает синтаксис оператора GRANT. Его обобщенный упрощенный вид представлен ниже:

```
DENY { ALL [ PRIVILEGES ] }
      | название_привилегии [ ( название_поля [ ,...n ]
) ] [ ,...n ]
[ ON [ класс_защищаемого_объекта :: ] защищаемый_объект ]
TO имя_учетной_записи [ ,...n ]
[ CASCADE ]
[ AS имя_учетной_записи]
```

Как видите, почти все параметры аналогичны оператору раздачи прав. Новым параметром является только **CASCADE**, который выполняет обратное действие по отношению к опции WITH GRANT OPTION инструкции GRANT. Он позволяет запретить права указанному пользователю и всем другим пользователям, которым они были предоставлены с помощью опции WITH GRANT OPTION. При этом, если забрать права у пользователя, которому эта привилегия была предоставлена с помощью опции WITH GRANT OPTION, и не указать аргумент CASCADE, SQL Server сгенерирует ошибку.

Например, предоставим всем пользователям, кроме пользователя Verhiv право просматривать информацию о магазинах, которые реализуют книги издательства:

```
grant select on book.Books to public;      --
предоставляем право select всем пользователям
deny select on book.Books to Verhiv;      -- отменяем право
select у пользователя Verhiv
```

Чтобы забрать предоставленные права у пользователей, используется оператор **REVOKE**. По сути, данный оператор

позволяет отменить действия, совершенные ранее оператором GRANT или DENY.

Формат данной команды схож с форматом команды GRANT.

```
REVOKE [ GRANT OPTION FOR ]
      { ALL [ PRIVILEGES ]
        | название_привилегии [ ( название_поля [ ,...n
] ) ] [ ,...n ]
      }
[ ON [ класс_защищаемого_объекта :: ] защищаемый_объект ]
{ TO | FROM } имя_учетной_записи [ ,...n ]
[ CASCADE ]
[ AS имя_учетной_записи]
```

Опция **GRANT OPTION FOR** позволяет забрать права на раздачу прав пользователем. При использовании параметра CASCADE данную опцию обязательно следует включать.

Параметр **CASCADE** указывает на каскадную отмену прав, которые предоставлялись с помощью параметра WITH GRANT OPTION оператора GRANT.

Например, чтобы забрать права на чтение из таблицы sale.Sales у пользователя Kovalenko нужно написать следующее:

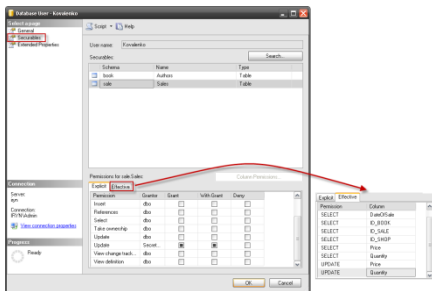
```
revoke select
on sale.Sales
from Kovalenko;
```

Если необходимо забрать права у нескольких пользователей, тогда их следует перечислить через запятую. А для аннулирования прав у всех пользователей укажите роль PUBLIC.

В следующем примере заберем право на раздачу привилегия select на поля Price и Quantity таблицы sale.Sale у пользователя Kovalenko.

```
revoke grant option for update
on sale.Sales(Price, Quantity)
from Kovalenko;
```

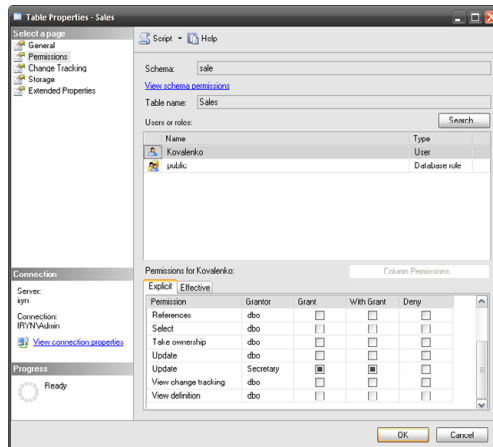
Для предоставления или удаления прав SSMS имеет также свой собственный, довольно удобный в использовании, набор средств. Например, для того, чтобы пересмотреть и изменить набор прав определенного пользователя или роли базы данных, необходимо перейти к их свойствам (Properties) и выбрать вкладку **Securables (Защита)**. Аналогичная вкладка существует при создании отдельно взятого пользователя (или роли базы данных). Выглядит она так:



Данный инструмент позволяет просмотреть, на какие объекты базы данных имеет права пользователь. В случае необходимости, новый объект можно добавить с помощью кнопки **Search ... (Поиск)**. В части **Permissions for (Права доступа для)** подробнее представлено, какие именно права имеет пользователь на выбранный объект базы данных.

На вкладке **Explicit** можно просмотреть и настроить явные права для пользователя. А на вкладке **Effective** – итоговые права доступа для пользователя или роли базы данных (поскольку права от различных ролей базы данных, назначенных этому пользователю, суммируются).

Получить информацию о том, кто и какие права имеет на отдельно взятый объект, можно с помощью вкладки **Permissions** свойств этого объекта. Например, для таблиц она имеет следующий вид:



Список **Users or roles (Пользователи или роли)** позволяет легко переключаться между пользователями или ролями базы данных, которые имеют права на эту таблицу. В части **Permissions for (Права доступа для)** вы можете более подробно ознакомиться с набором прав текущего (выделенного) пользователя или роли базы данных.

Наконец отметим, что на практике используется более десятка таблиц и других объектов базы данных, доступ к которым предоставлять каждому пользователю довольно

неудобно. Упростить эту задачу помогает установления прав доступа на уровне схемы или всей базы данных, а также использование фиксированных ролей уровня базы данных.

Также не забывайте о том, что доступ пользователей на прямую к таблицам организовывать не стоит. Зато изменении данных лучше организовать с помощью хранимых процедур, а для просмотра данных использовать представления или те же хранимые процедуры.

5.3. Командные права доступа

В SQL Server пользователь, кроме объектных прав, может иметь также набор командных прав, позволяющих ему выполнять определенный набор операторов (команд). Фактически – это набор прав на базу данных. В SQL Server 2008 список командных прав достаточно большой и имеет следующий вид:

- ALTER;
- ALTER ANY APPLICATION ROLE;
- ALTER ANY ASSEMBLY;
- ALTER ANY ASYMMETRIC KEY;
- ALTER ANY CERTIFICATE;
- ALTER ANY CONTRACT;
- ALTER ANY DATABASE AUDIT;
- ALTER ANY DATABASE DDL TRIGGER;
- ALTER ANY DATABASE EVENT NOTIFICATION;
- ALTER ANY DATASPACE;
- ALTER ANY FULLTEXT CATALOG;
- ALTER ANY MESSAGE TYPE;
- ALTER ANY REMOTE SERVICE BINDING;

- ALTER ANY ROLE;
- ALTER ANY ROUTE;
- ALTER ANY SCHEMA;
- ALTER ANY SERVICE;
- ALTER ANY SYMMETRIC KEY;
- ALTER ANY USER;
- AUTHENTICATE;
- BACKUP DATABASE;
- BACKUP LOG;
- CHECKPOINT;
- CONNECT;
- CONNECT REPLICATION;
- CONTROL;
- CREATE AGGREGATE;
- CREATE ASSEMBLY;
- CREATE ASYMMETRIC KEY;
- CREATE CERTIFICATE;
- CREATE CONTRACT;
- CREATE DATABASE;
- CREATE DATABASE DDL EVENT NOTIFICATION;
- CREATE DEFAULT;
- CREATE FULLTEXT CATALOG;
- CREATE FUNCTION;
- CREATE MESSAGE TYPE;
- CREATE PROCEDURE;
- CREATE QUEUE;
- CREATE REMOTE SERVICE BINDING;
- CREATE ROLE;
- CREATE ROUTE;
- CREATE RULE;

- CREATE SCHEMA;
- CREATE SERVICE;
- CREATE SYMMETRIC KEY;
- CREATE SYNONYM;
- CREATE TABLE;
- CREATE TYPE;
- CREATE VIEW;
- CREATE XML SCHEMA COLLECTION;
- DELETE;
- EXECUTE;
- INSERT;
- REFERENCES;
- SELECT;
- SHOWPLAN;
- SUBSCRIBE QUERY NOTIFICATIONS;
- TAKE OWNERSHIP;
- UPDATE;
- VIEW DATABASE STATE;
- VIEW DEFINITION.

Управление командными правами ничем особым от управления объектными не отличается. Для этого также используются операторы GRANT, DENY и REVOKE, но несколько в иной форме.

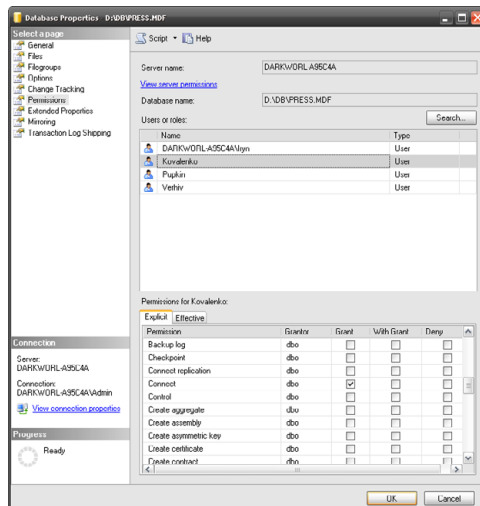
```
-- раздача прав
GRANT { ALL [ PRIVILEGES ] | название_привилегии [
, ...n ] }
TO имя_учетной_записи [ , ...n ]
[ WITH GRANT OPTION ]
[ AS имя_учетной_записи]
```

```
-- аннулировать или забрать права
{ DENY | REVOKE } { ALL [ PRIVILEGES ] | название_
привилегии [ ,...n ] }
TO имя_учетной_записи [ ,...n ]
[ CASCADE ]
[ AS имя_учетной_записи]
```

Приведем примеры управления следующими правами:

```
-- предоставим права пользователю Kovalenko на создание
таблицы
grant create table
to Kovalenko;
-- заберем в роли Management права на создание
представления
revoke create view
to Management;
```

Чтобы настроить командные права доступа к базе данных с помощью SSMS, необходимо выбрать в контекстном меню базы данных пункт **Properties (Свойства)** и перейти на вкладку **Permissions (Права доступа)**.



Этот графический интерфейс позволяет раздавать или аннулировать избранное право для текущего пользователя или роли базы данных, путем простого расставления или снятия галочек в части **Permissions for (Права доступа для)**.

5.4. Данные про права

Для получения информации о том, кто и какие права имеет, кроме уже рассмотренных способов (из свойств отдельно взятого объекта), можно воспользоваться еще системной хранимой процедурой **sp_helprotect**. Она позволяет просмотреть информацию о правах доступа к объектам текущей базы данных.

```
sp_helprotect [ [ @name = ] 'оператор_или_объект_БД' ]
-- информацию о каких правах

- отразить
    [ , [ @username = ] 'имя_учетной_записи' ]
-- права либо участника проверяются
    [ , [ @grantorname = ] 'имя_учетной_записи' ] --
каким пользователям были предоставлены права
    [ , [ @permissionarea = ] 'тип' ] -- тип
прав, которые отражаются
```

Как видно из синтаксиса, аргументы хранимой процедуры являются необязательными. Таким образом, если ее запустить на выполнение без аргументов, на экран будет выведен список всех прав, которые были предоставлены или запрещены для текущей базе данных.

Замечания по аргументам хранимой процедуры:

1. Если аргумент **name** является оператором, тогда он может принимать только одно из следующих значений:
 - CREATE DATABASE;
 - CREATE DEFAULT;
 - CREATE FUNCTION;
 - CREATE PROCEDURE;
 - CREATE RULE;
 - CREATE TABLE;
 - CREATE VIEW;
 - BACKUP DATABASE;
 - BACKUP LOG.
2. Аргумент **permissionarea** указывает на то, какие права будут отражены: на объекты (символ "o"), операторы (символ "s") или на то и другое ("os", значение по умолчанию). При использовании комбинаций типов, символы "o" и "s" разделяются запятой или пробелом.
3. Если необходимо указать только несколько аргументов, вместо пропущенных следует указывать значение NULL или использовать именованные аргументы. Например, необходимо пересмотреть все права, которые имеет пользователь Kovalenko:

```
exec sp_helprotect NULL, Kovalenko
-- или
exec sp_helprotect @username = 'Kovalenko'
```

Результат:

Results		Messages					
	Owner	Object	Grantee	Grantor	ProtectType	Action	Column
1	book	Authors	Kovalenko	Secretary	Grant	Select	(All+New)
2	sale	Sales	Kovalenko	Secretary	Grant_WGO	Update	Price
3	sale	Sales	Kovalenko	Secretary	Grant_WGO	Update	Quantity
4	.	.	Kovalenko	dbo	Grant	CONNECT	.

Данные в результирующем множестве сортируются по категории прав, владельцу, объекта, получателю прав, участнику, который предоставил привилегию, категории типа защиты, типа защиты, действия и по полю идентификатора.

Приведем еще один пример использования хранимой процедуры `sp_helprotect`. Выведем список прав на таблицу `sale.Sales`:

```
exec sp_helprotect 'sale.Sales'
```

Результат:

Results		Messages					
	Owner	Object	Grantee	Grantor	ProtectType	Action	Column
1	sale	Sales	Kovalenko	Secretary	Grant_WGO	Update	Price
2	sale	Sales	Kovalenko	Secretary	Grant_WGO	Update	Quantity
3	sale	Sales	public	Secretary	Grant	Select	(All+New)

Поскольку рассмотренная хранимая процедура не возвращает информацию о защищаемых объектах, которые появились еще в SQL Server 2005, для таких целей рекомендуется использовать представления **`sys.database_permissions`** и функцию **`fn_builtin_permissions`**. Более подробно о них читайте в документации по SQL Server 2008.

