Администрирование SQL Server

Часть 1. Безопасность

1.1. Пользователи

Для того чтобы получить доступ к какой-либо из баз данных, следует вначале создать учетную запись.

В случае Windows-аутентификации учетная запись берется из списка локальных пользователей компьютера или из списка пользователей домена.

Учетной записью может быть и группа пользователей. *Это бывает очень удобно, так как одним действием предоставляется доступ к серверу целой группе пользователей.*

1.1. Пользователи

Если предполагается смешанная аутентификация (когда наряду с Windows-аутентификацией возможно задание логина и пароля определенного пользователя прямо в SQL Server), то следует указать имя учетной записи и ее пароль.

Для учетной записи можно указать одну из девяти встроенных ролей, которые определяют возможность выполнять определенный набор действий на уровне сервера.

1.1. Пользователи

После создания учетной записи следует создать пользователя базы данных, доступ к которой будет разрешен данной учетной записи, и увязать пользователя с этой учетной записью.

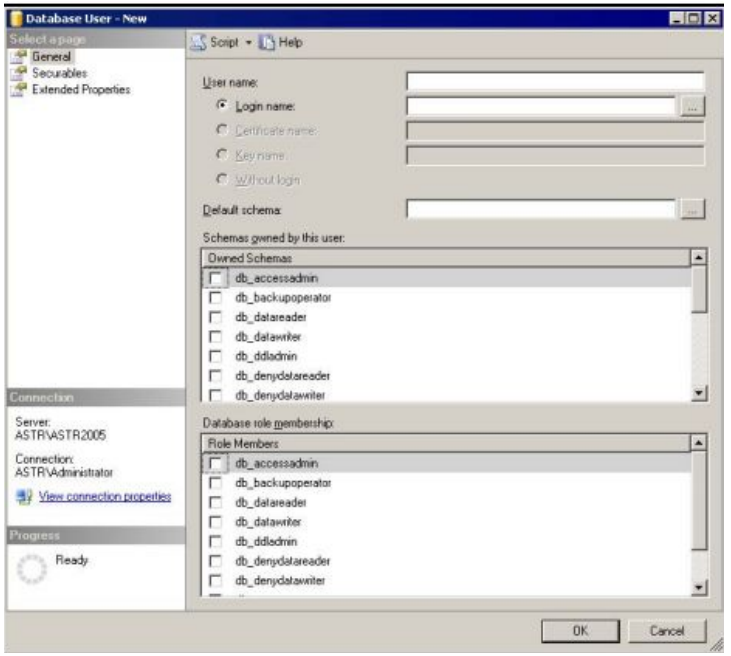
После создания пользователя можно указать, что пользователь будет являться членом одной из ролей базы данных.

1.1. Пользователи

Также можно создать пользователя с помощью SQL Server Management Studio.

Для этого необходимо открыть список пользователей базы данных, выбрав Security | Users, а потом щелкнуть правой кнопкой мыши на этом списке и выбрать пункт New User.

Появится окно (см. рисунок справа).



Далее следует ввести имя пользователя, его роль и выбрать Windows-пользователя для него.

Затем нажать на кнопку OK — и пользователь базы данных будет создан.

1.2. Роли

Роль в SQL Server похожа на группу пользователей.

Роль позволяет объединять пользователей, выполняющих одинаковые функции (т. е. играющие одинаковые роли), для упрощения администрирования SQL Server.

Существуют:

- ✔ роли уровня сервера
- ✔ роли уровня базы данных.

1.2. Роли

При установке на уровне сервера создается 8 предопределенных фиксированных ролей.

При создании базы данных также создается 10 предопределенных фиксированных ролей.

1.2. Роли

Фиксированные роли на уровне сервера:

sysadmin — разрешено выполнять любые действия в SQL Server;
setupadmin — управление связанными серверами;
serveradmin — доступно конфигурирование и выключение сервера;
securityadmin — управляет учетными записями и правами на создание базы данных;
processadmin — может управлять процессами, запущенными в SQL Server;
diskadmin — управляет файлами SQL Server;
dbcreator — разрешено создавать и модифицировать базы данных;
bulkadmin — может управлять процессом массовой вставки данных.

1.2. Роли

Фиксированные роли базы данных:

db_accessadmin — может удалять или добавлять пользователей базы данных;
db_backupoperator — может выполнять операцию резервного копирования;
db datareader — может просматривать содержимое таблиц базы данных;
db_datawriter — может изменять данные таблиц;
db_ddladmin — может выполнять команды DDL (команды SQL для создания таблиц и структур баз данных);
db_denydatareader — запрещается просматривать данные в любой таблице;
db_denydatawriter — запрещается модифицировать данные в любой таблице;
db_owner — обладает всеми правами в базе данных (собственник базы данных);
db_securityadmin — может управлять ролями и разрешениями на уровне базы данных;
public — любой пользователь базы данных автоматически становится членом этой роли.
Данную роль используют для предоставления минимальных прав пользователю.

1.2. Роли

Кроме использования встроенных ролей, можно создавать свои роли, причем как для пользователей, так и для приложений.

Можно создать роль с помощью SQL Server Management Studio.

Для каждого пользователя базы данных и для каждой роли можно явно прописать права и запреты для конкретных объектов базы данных.

Можно прописывать права и ограничения даже на уровне отдельных столбцов таблицы. Но гораздо проще и лучше использовать **схемы** для этих целей.

1.3. Схемы

Небольшое отступление.

Существует концепция групп и ролей, служащих для управления пользователями в Windows.

На пользователя, относящегося к определенной группе, распространяются все права и ограничения этой группы.

Схема — это средство группировки объектов (не пользователей), позволяющее обращаться с набором объектов как с единым целым (в смысле владения и прав).

Пример.

Можно при помощи одного оператора T-SQL предоставить роли право на выполнение всех хранимых процедур, относящихся к определенной схеме.

Это очень удобно, если в системе сотни хранимых процедур.

1.3. Схемы

Некоторая информация о схемах:

- для каждого пользователя базы данных определяется своя схема по умолчанию. При создании пользователем объекта базы данных этот объект по умолчанию помещается в данную схему;

- каждый объект базы данных, за исключением объектов безопасности, относится к какой-либо схеме;

1.3. Схемы

Некоторая информация о схемах:

- при создании объекта можно явно указать схему, в которую его нужно поместить.

  В базе данных могут существовать объекты с одинаковыми именами, но принадлежащие разным схемам.

  Но одинаковые имена объектов в одной схеме не допускаются;

1.3. Схемы

Некоторая информация о схемах:

- при создании объекта можно явно указать схему, в которую его нужно поместить.

  В базе данных могут существовать объекты с одинаковыми именами, но принадлежащие разным схемам.

  Но одинаковые имена объектов в одной схеме не допускаются;

- для пользователей и ролей, для облегчения администрирования указываются определенные права на объекты той или иной схемы.

1.3. Схемы

Отделение пользователей от схем, начиная с SQL Server 2005, дает некоторые преимущества:

- более легкое удаление пользователей. Пользователь не владеет объектами базы данных.

  Если в SQL Server 2000 пользователь владел каким-либо объектом базы, и его необходимо удалить, то надо было сначала передать эти объекты во владение другому пользователю, что сильно усложняло процесс удаления.

- схемой (и ее объектами) могут управлять несколько пользователей.

1.4. Использование схемы

Перед использованием схемы ее нужно создать, что можно сделать с помощью оператора CREATE SCHEMA:

```
CREATE SCHEMA MySchema
```

Этот оператор создает схему MySchema, которой владеет пользователь, выполняющий данный оператор.

1.4. Использование схемы

После создания схемы ее можно использовать двумя способами.

Первый: он просто требует, чтобы при создании объектов в БД указывали имя схемы.

Например, следующий код создает в схеме MySchema таблицу MyNewTable.

При создании этой таблицы используется явная нотация схема.объект.

```
CREATE TABLE MySchema.MyNewTable (ID int, Name nvarchar(100))
```

1.4. Использование схемы

Второй путь основан на использовании схемы по умолчанию и потому не требует указания схемы в операторе создания объекта.

Пример 1. Создание объекта:

```
CREATE TABLE MyNewTable (ID int, Name nvarchar(100))
```

В этом примере таблица создается в схеме по умолчанию, соответствующей пользователю, который выполняет данный оператор.

Например, если этот код выполняет пользователь Alex и его схемой по умолчанию является схема Staff, то таблица создается в схеме Staff и владельцем объекта будет схема Staff.

1.4. Использование схемы

ВНИМАНИЕ!

В SQL Server 2000 не было различия между пользователями и схемами, поэтому "схемой по умолчанию" был пользователь.

Начиная с SQL Server 2005, пользователи отделены от схем, поэтому можно связать нескольких пользователей с одной схемой по умолчанию.

Лучше всегда создавать объекты явно — это исключит многие потенциальные затруднения.

1.5. Доступ к объектам

Если пользователю назначена схема по умолчанию, то при ссылке на объект из этой схемы ее имя указывать не нужно.

Если есть разные схемы, то при обращении к объекту без указания его схемы порядок действий таков:

1. SQL Server проверяет, существует ли объект в схеме sys. Если да, то он использует этот объект.
2. SQL Server проверяет, существует ли объект в назначенной пользователю схеме по умолчанию. Если да, то он использует этот объект.
3. SQL Server проверяет, существует ли объект в схеме dbo.
   Если да, то он использует этот объект.

1.5. Доступ к объектам

Пример 2.

Если таблица Product существует в схемах Production и dbo:

1. Проверяется, есть ли таблица Product в схеме sys.
   Если нет, проверка продолжается.

2. Проверяется, есть ли таблица Product в схеме Staff.
   Если нет, проверка продолжается.

3. Проверяется, есть ли таблица Product в схеме dbo.
   Если да, используется найденная таблица.

1.5. Доступ к объектам

Пример 3.

Если происходит запрос к таблице Employee, которая существует только в схеме Company, то происходит следующее:

1. Проверяется, есть ли таблица Employee в схеме sys.
   Если нет, проверка продолжается.

2. Есть ли таблица Employee в схеме Staff.
   Если нет, проверка продолжается.

3. Есть ли таблица Employee в схеме dbo.
   Если нет, происходит ошибка.

1.5. Доступ к объектам

Пример 3.

Если происходит запрос к таблице Employee, которая существует только в схеме Company, то происходит следующее:

1. Проверяется, есть ли таблица Employee в схеме sys.
   Если нет, проверка продолжается.

2. Есть ли таблица Employee в схеме Staff.
   Если нет, проверка продолжается.

3. Есть ли таблица Employee в схеме dbo.
   Если нет, происходит ошибка.

В примере №3 схемой по умолчанию для пользователя Alex является схема Staff, а таблица Employee существует только в схеме Company.

Поэтому в запросе необходимо было явно указать, к какой схеме относится таблица.

Чтобы у Вас не возникали вопросы о том, к какому объекту Вы обращаетесь на самом деле, при наличии нескольких схем придерживайтесь явной методики обращения к объектам базы данных!

1.6. Права

Если в SQL Server 2000 необходимо было предоставить владельцу хранимых процедур право на их выполнение, а также позволить ему выполнять выборку данных из принадлежащих ему таблиц, то надо было назначить роли права в отношении каждого объекта.

1.6. Права

Пример 4.

Так это было бы в SQL Server 2000 :

```
GRANT EXECUTE ON MyProcedure TO MyDatabaseRole
GRANT SELECT ON MyTable1 TO MyDatabaseRole
```

При большом количестве таблиц и процедур этот способ становится очень громоздким, а риск ошибиться — очень высоким.

Но начиная с SQL Server 2005, можно сгруппировать объекты в схему и назначить права самой схеме.

1.6. Права

Пример 5.

Оператор GRANT предоставляет пользователю Alex право на выборку данных из всех объектов:

```
GRANT SELECT, EXECUTE ON Schema::Sales TO Alex
```

Эта новая возможность появилась, начиная с SQL Server 2005.

Кроме того, как и в предыдущих версиях, SQL Server позволяет назначать права и ролям, только теперь Вы можете сделать это на уровне схемы:

```
GRANT SELECT, EXECUTE ON Schema::Sales TO MyDatabaseRole
```

С помощью одного простого оператора Вы можете назначить группе пользователей все необходимые права в отношении набора объектов БД.

1.7. Синонимы

Начиная с SQL Server 2005, появились синонимы, позволяющие создавать постоянные псевдонимы для объектов базы данных.

Пример 6.
Имеется представление:

```
CREATE VIEW MyView
SELECT ProductID, ProductName
FROM Production.Product
```

1.7. Синонимы

```
CREATE VIEW MyView
SELECT ProductID, ProductName
FROM Production.Product
```

В этом примере создается представление, использующее таблицу Product из схемы Production.

Можно создать синоним для этой таблицы, чтобы было проще к ней обращаться:

```
CREATE SYNONYM sProduct FOR Production.Product
```

Теперь можно заменить обращение к таблице обращением к синониму, как показано:

```
ALTER VIEW MyView
SELECT ProductID, ProductName
FROM sProduct
```

1.8. Конструкция EXECUTE AS

Эта конструкция появилась еще в SQL Server 2005.

Она позволяет изменять контекст безопасности, в котором выполняется команда.

Конечно, пользователь, желающий выполнить метод, должен иметь право на его выполнение.

В начале выполнения метода контекст безопасности изменяется на новый контекст, определяемый конструкцией EXECUTE AS.

1.8. Конструкция EXECUTE AS

Конструкцию EXECUTE AS можно использовать с хранимыми процедурами и пользовательскими функциями (за исключением встраиваемых табличных функций).

1.8. Конструкция EXECUTE AS

Пример 7.

Допустим, что Alex имеет право на выполнение хранимой процедуры Sales.GetOrderList и выполняет ее. Схемой Sales владеет Ann.

Эта процедура обращается к ряду таблиц, относящихся к нескольким разным схемам.

Всеми этими схемами владеет Ann.

Хотя Alex не имеет права на непосредственную выборку данных из этих таблиц, но благодаря цепочке владения и тому факту, что все нужные схемы принадлежат одному владельцу, процедура выполняется без проблем.

1.8. Конструкция EXECUTE AS

Однако, если одна из используемых в процедуре таблиц будет относиться к схеме, не принадлежащей Ann, цепочка владения оборвется, и для выборки данных из этой таблицы Alex должен будет получить дополнительные права.

1.9. Конструкция EXECUTE AS CALLER

При использовании этого варианта метод выполняется в контексте вызвавшего его пользователя.

EXECUTE AS CALLER соответствует также контексту выполнения по умолчанию, поэтому при создании хранимой процедуры или пользовательской функции эту конструкцию можно не указывать.

1.9. Конструкция EXECUTE AS CALLER

Пример 8.

Хранимая процедуры без заданного контекста выполнения:

```
CREATE PROCEDURE procProductDetails (@ProductID int)
AS
SELECT ProductID, ProductName
FROM Production.Product
WHERE ProductID=@ProductID
```

1.9. Конструкция EXECUTE AS CALLER

```
CREATE PROCEDURE procProductDetails (@ProductID int)
AS
SELECT ProductID, ProductName
FROM Production.Product
WHERE ProductID=@ProductID
```

Отсутствие конструкции EXECUTE AS CALLER идентично ее наличию:

```
CREATE PROCEDURE procProductDetails (@ProductID int)
WITH EXECUTE AS CALLER
AS
SELECT ProductID, ProductName
FROM Production.Product
WHERE ProductID=@ProductID
```

Если Вы хотите, чтобы метод выполнялся в контексте вызвавшего пользователя, вы можете включить в код конструкцию EXECUTE AS CALLER или просто опустить ее.

1.10. Конструкция EXECUTE AS в контексте пользователя

Этот вариант конструкции EXECUTE AS позволяет выполнить метод в контексте заданного пользователя.

Пример 9.

```
CREATE PROCEDURE procProductDetails (@ProductID int)
WITH EXECUTE AS 'Ann'
AS
SELECT ProductID, ProductName
FROM Production.Product
WHERE ProductID=@ProductID
```

1.10. Конструкция EXECUTE AS в контексте пользователя

```
CREATE PROCEDURE procProductDetails (@ProductID int)
WITH EXECUTE AS 'Ann'
AS
SELECT ProductID, ProductName
FROM Production.Product
WHERE ProductID=@ProductID
```

Когда кто-то вызывает эту процедуру, она выполняется так, как если бы ее вызвал пользователь Ann.

Если в коде процедуры осуществляется доступ к каким-либо другим объектам базы данных, то все будет работать так, как если бы к ним пыталась обратиться Ann, при этом используются права Ann.

1.10. Конструкция EXECUTE AS в контексте пользователя

```
CREATE PROCEDURE procProductDetails (@ProductID int)
WITH EXECUTE AS 'Ann'
AS
SELECT ProductID, ProductName
FROM Production.Product
WHERE ProductID=@ProductID
```

Например, если Ann имеет право на выборку данных из таблицы Product, то при вызове процедуры пользователем Alex доступ к таблице Product будет выполнен так, как если бы к ней обратилась Ann.

1.10. Конструкция EXECUTE AS в контексте пользователя

```
CREATE PROCEDURE procProductDetails (@ProductID int)
WITH EXECUTE AS 'Ann'
AS
SELECT ProductID, ProductName
FROM Production.Product
WHERE ProductID=@ProductID
```

Без конструкции EXECUTE AS пользователю, выполняющему эту процедуру, потребовалось бы дополнительное право на выборку данных из таблицы Product, но так как обращение к этой таблице выполняется в контексте пользователя Ann, обрыва цепочки владения не происходит.

1.10. Конструкция EXECUTE AS в контексте пользователя

```
CREATE PROCEDURE procProductDetails (@ProductID int)
WITH EXECUTE AS 'Ann'
AS
SELECT ProductID, ProductName
FROM Production.Product
WHERE ProductID=@ProductID
```

Пока Ann принадлежит право на выборку данных из таблицы Product, любой пользователь, имеющий право на выполнение этой хранимой процедуры, может использовать ее без каких-либо нарушений защиты.