Урок 9

План заняття:

1. Системні таблиці та представлення

Важливо знати, що кожна база даних, крім створених користувачем таблиць, містить додатковий набір системних таблиць. Імена більшості системних таблиць починаються з префікса **sys**. Більшість полів цих таблиць документовані, але трапляються і не документовані поля, звернення до яких здійснювати заборонено.

Отже, давайте розберемось, які системні таблиці існують і яка інформація в них представлена. Всі системні таблиці можна поділити на **наступні групи**:

1. Таблиці резервного копіювання та відновлення.
2. Таблиці доставки журналів. Їх імена починаються з префікса **log_shipping_xxx**.
3. Таблиці системи відслідковування змін даних.
4. Таблиці реплікації.
5. Таблиці плану обслуговування бази даних.
6. Таблиці агента SQL Server.
7. Таблиці служб Integration Services, які фактично містять опис системних таблиць бази даних msdb.
8. Базові системні таблиці, які зберігають метадані окремої бази даних.

Отже, набір системних таблиць доволі великий. Основну увагу привертають базові системні таблиці. Вони розміщуються в базах даних користувачів та в головній базі даних master. Але дані базових системних таблиць закриті і недоступні для звичайних клієнтів SQL Server. Доступ до них мають лише співробітники корпорації Microsoft.

Існує ще одна база даних, яка містить перелік всіх системних об'єктів, які входять до складу SQL Server, тобто включають також в себе набір базових системних таблиць користувацьких баз даних. Це база даних **Resource**. Системні об'єкти SQL Server, такі як sys.objects, фізично зберігаються в базі даних Resource, а відображаються у схемі **sys** для кожної бази даних (оскільки сама база даних Resource невидима і доступ напряму до неї заборонений).

Варто відмітити, що ряд системних таблиць, які були присутні в попередніх версіях SQL Server, починаючи з версії MS SQL Server 2005 реалізовані у вигляді набору системних представлень. Ці системні представлення несуть загальну інформацію і реалізовані на серверному рівні, тобто можуть розповісти про будь-яку базу даних на сервері. Ряд цих представлень ви вже знаєте, оскільки про них ми згадували в попередніх уроках. Та не дивлячись на це, підсумуємо всю

відому інформацію та приведемо співставлення системних таблиць SQL Server 2000 (остання версія, де використовувались приведені системні таблиці) та системних представлень SQL Server 2008.

Спочатку розглянемо системні представлення, які є в кожній базі даних на сервері.

Системна таблиця SQL Server 2000	Системне представлення SQL Server 2008	Опис
sysobjects	sys.objects	Містить перелік об'єктів бази даних, крім тригерів.
syscomments	sys.sql_modules	Містить код оголошення об'єктів бази даних (модулів) та різноманітні параметри, з якими створювався кожен окремий об'єкт.
syscolumns	sys.columns	Містить загальний опис кожного поля таблиці, представлення та повертаємого табличне значення функції.
sysdepends	sys.sql_expression_dependencies	Містить дані про іменовані залежності між представленнями, зберігаємими процедурами, тригерами і таблицями, які вказуються при їх оголошенні.
sysfilegroups	sys.filegroups	Містить інформацію про файлові групи, які є в базі даних.
sysfiles	sys.database_files	Містить дані про файли бази даних.
sysaforeignkeys	sys.foreign_key_columns	Зберігає інформацію про всі зовнішні ключі, включаючи дані про те, з первинним полем якої таблиці він зв'язаний.
sysconstraints	sys.check_constraints sys.default_constraints sys.key_constraints sys.foreign_keys	Містить інформацію про конструкції бази даних: обмеження (check), значення по замовчуванню (default), ключі (key), включаючи зовнішні (foreign keys).
sysindexes	sys.indexes sys.partitions sys.allocation_units sys.dm_db_partition_stats	Містять інформацію про всі індекси бази даних.
sysindexkeys	sys.index_columns	Містить інформацію про індекси і поля, на які вони вказують.
sysmembers	sys.database_role_members	Містить інформацію про користувачів і ролі, до яких вони відносяться.
sysreferences	sys.foreign_keys	Містить карту зв'язків таблиць.
systypes	sys.types	Містить список всіх користувацьких та системних типів даних.
sysusers	sys.database_principals	Містить дані про всіх користувачів та ролі SQL Server і Windows.
sysfulltextcatalogs	sys.fulltext_catalogs	Містить список повнотекстових каталогів.
syspermissions sysprotects	sys.database_permissions sys.server_permissions	Містить дані про всі права і обмеження окремого користувача або ролі: на рівні поля бази даних (sys.database_permissions) або на рівні сервера (sys.server_permissions).

Ряд системних таблиць бази даних master в SQL Server 2008 також були замінені на системні представлення з аналогічними іменами. Наприклад, системна таблиця **sysconfigures**, яка містить параметри конфігурації сервера в системі, замінена на системне представлення **sys.configurations**, системна таблиця **sysdatabases** з інформацією про перелік баз даних сервера – на системне представлення **sys.databases** тощо.

2. Захист даних. Режими захисту даних. Поняття аутентифікації та авторизації на рівні MS SQL Server

2.1. Основні поняття

Сьогодні ми навчимось належним чином організовувати безпеку даних в SQL Server 2008, яка має важливе значення в діяльності різного роду організацій, що використовують бази даних. Адже без забезпечення надійного захисту даних, будь-хто може знищити цінні дані або отримати доступ до засекреченої інформації. Для цього в першу чергу слід вірно спроектувати систему управління користувачами, зокрема зменшити кількість користувачів з необмеженим доступом до мінімуму, ввести систему паролів тощо.

Система безпеки SQL Server 2008 для запобігання неавторизованого доступу до даних використовує режими захисту та облікові записи. Існує **2 режими захисту даних**:

1. **Режим захисту Windows (Windows authentication)** – це самий простий спосіб захисту і він являється режимом захисту даних по замовчуванню. Цей режим аутентифікації дозволяє підключатись до SQL Server, вводячи логін та пароль при вході в Windows. Щоб задіяти даний режим, достатньо на сервері додати ім'я входу Windows.
2. **Змішаний режим захисту (Mixed mode)** – це режим захисту, при якому користувачі реєструються безпосередньо при доступі до SQL Server, окремо від реєстрації в операційній системі. При цьому доступ до даних можуть отримувати будь-які імена входу, включаючи імена входу Windows. Зазвичай, даний спосіб аутентифікації обирають у випадку використання відмінної від Windows операційної системи.

Корпорація Microsoft рекомендує використовувати режим захисту Windows, оскільки він дозволяє реалізувати переваги всіх централізованих політик безпеки домена Active Directory. Такий режим дійсно має ряд **переваг** для виробничої бази даних, серед яких можна виділити наступні:

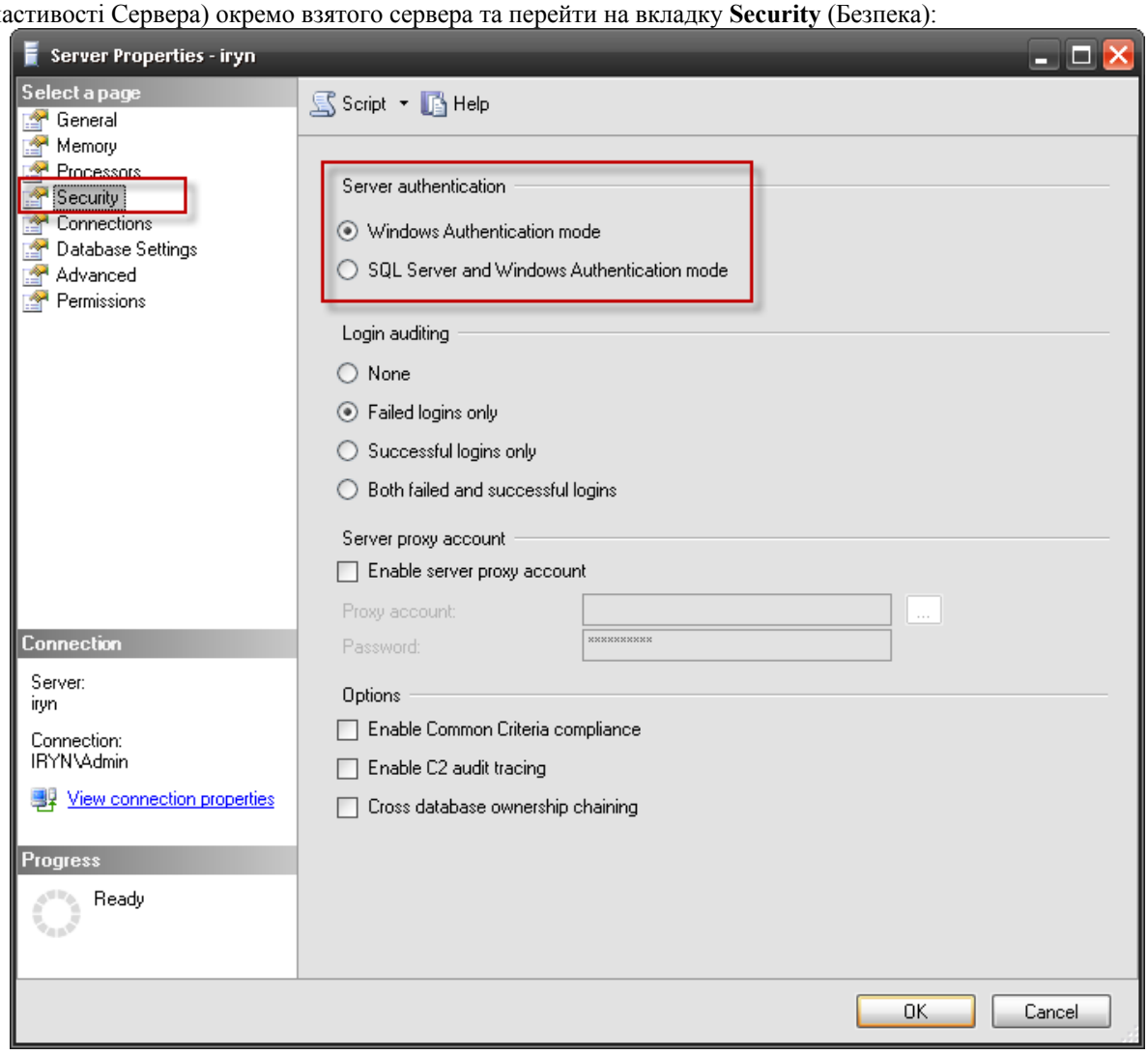
- ✓ користувачу не потрібно запам'ятовувати багато паролів, а достатньо одного – для входу на комп'ютер;
- ✓ перевірка даних користувача при вході на сервер відбувається швидше.

Але вибір на користь змішаного режиму може також переважити:

- на підприємстві не існує домена Windows, наприклад, при використанні Unix систем;
- користувачі SQL Server не входять в домен, наприклад, якщо вони підключаються до сервера баз даних через Web-інтерфейс.

Підсумуємо: режим захисту Windows найбільш безпечний, але змішаний режим захисту більш гнучкий і незалежний від адміністратора мережі.

Режим захисту даних можна змінити. Для цього в SSMS необхідно викликати діалогове вікно **Server Properties** (Властивості Сервера) окремо взятого сервера та перейти на вкладку **Security** (Безпека):



Крім того, в системі SQL Server організована **двохрівнева модель** обмеження доступу до даних. В зв'язку з цим **порядок дій організації доступу до даних користувача** буде наступний:

1. Створити обліковий запис або ім'я входу користувача (login) на сервері, вказавши для нього пароль та набір інших параметрів. Це дозволить йому підключатись до самого сервера, але ще не дає доступу до баз даних. Даний етап являється першим рівнем безпеки SQL Server **(аутентифікація користувача)**.
2. До кожної необхідної бази даних додати користувача (user), який буде співставлятись з іменем входу. На основі прав користувача баз даних (user), його ім'я входу (login) отримує доступ до відповідної бази даних. В зв'язку з цим на одне ім'я входу може припадати кілька користувачів (для різних баз даних), кожен з яких може мати різні права доступу. Це другий рівень безпеки **(авторизація користувача).**
3. Надати користувачу відповідні права на базу даних.

Розглянемо все по порядку.

2.2. Створення імен входу

Отже, для організації безпеки на сервері баз даних необхідно в першу чергу правильно спроектувати систему управління користувачами. В мові SQL немає набору стандартних команд, які призначені для створення імен входу та користувачів бази даних. В кожній СУБД для цього призначений свій набір команд, але їх суть у всіх реалізаціях однакова. В SQL Server 2008 для таких цілей можна скористатись засобами SSMS або набором операторів T-SQL.

В T-SQL для створення імен входу (облікових записів, логінів) на сервері використовують оператор **CREATE LOGIN**.

```
CREATE LOGIN логін { WITH список_опцій | FROM джерело_даних }

-- Список опцій:
PASSWORD = { 'пароль' | 'хешируваний_пароль' HASHED } [ MUST_CHANGE ]
[, SID = sid ]                                    -- ідентифікатор безпеки; GUID нового
                                                  -- імені входу SQL Server
[, DEFAULT_DATABASE = БД_по_замовчуванню ]        -- база даних, яка буде поточною
                                                  -- при вході користувача
[, DEFAULT_LANGUAGE = мова_по_замовчуванню ]
[, CHECK_EXPIRATION = { ON | OFF} ] -- чи необхідно встановлювати строк дії пароля
[, CHECK_POLICY = { ON | OFF} ]     -- чи застосовувати до логіна політику паролів
                                    -- Windows на комп'ютері, де виконується SQL Server
[, CREDENTIAL = ім'я_облікових_даних ]

-- Джерело даних:
-- для Windows аутентифікації
WINDOWS [ WITH [ DEFAULT_DATABASE = БД_по_замовчуванню ]
               [, DEFAULT_LANGUAGE = мова_по_замовчуванню] ]
| CERTIFICATE ім'я_сертифіката
| ASYMMETRIC KEY ім'я_асиметричного_ключа
```

При створенні логіна спочатку потрібно вказати його ім'я, яке може містити від 1 до 128 символів (літери, цифри і всі символи, крім символа зворотнього слеша (\ )). В якості імені входу не можна використовувати зарезервовані імена, а їх значення не може бути NULL або являтись пустим рядком ('). Слід відмітити, що в SQL Server 2008 існує **4 типи імен входу**:

1) імена входу SQL Server;
2) імена входу Windows, які співставляються з обліковим записом домена Windows. Формат таких імен має вигляд [домен\логін];
3) імена входу, які співставлені за допомогою сертифіката;
4) імена входу, які співставлені за допомогою асиметричного ключа.

Для того, щоб створити ім'я входу SQL Server необхідно обрати конструкцію **WITH**. Для створення одного з трьох останніх типів підходить конструкція **FROM** оператора CREATE LOGIN.

Зупинимось коротко на поясненні останніх двох типів імен входу, а саме на іменах входу, які співставляються за допомогою сертифіката та асиметричного ключа. Почати напевно варто з того, що SQL Server 2008 підтримує ієрархічну структуру ключів, які дозволяють шифрувати дані. Сертифікати і асиметричні ключі являються одним із засобів такого шифрування. Вони повинні міститись в базі даних master і бути зв'язані з відповідним користувачем.

Асиметричний ключ (asymmetric key) – це комбінація закритого і відкритого ключа, який йому відповідає. Для його створення використовується оператор CREATE ASYMMETRIC KEY. **Сертифікати (certificates)** являють собою підписану цифровим підписом інструкцію, яка зв'язує значення відкритого ключа з ідентифікатором користувача (пристрою або служби), який має відповідний закритий ключ. В SQL Server 2008 можна створювати власні підписані сертифікати, які відповідають стандарту X.509 за допомогою інструкції CREATE SERTIFICATE. Сертифікати являються найнадійнішим способом шифрування даних, хоча разом з асиметричним ключем являється ресурсоємким і тому повільним методом.

Найшвидший метод шифрування даних – це **симетричний ключ**. Більш детальніше про конфігурування шифрування в SQL Server можна прочитати в розділі «Ієрархія засобів шифрування» електронної документації SQL Server 2008.

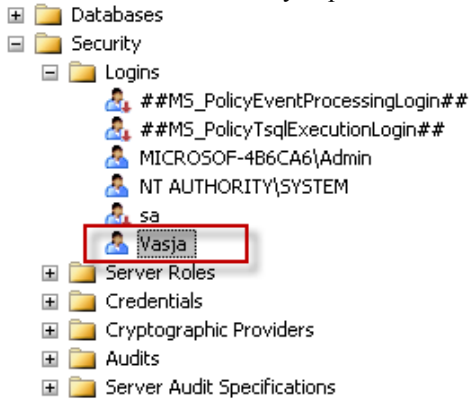
Зауваження щодо створення імен входу SQL Server:

- Якщо параметр **HASHED** не вказаний, тоді пароль хешується перед збереженням в базу даних. Даний параметр не рекомендується використовувати при створенні нових логінів. Він може бути корисний лише у конкретних випадках, наприклад, при перенесенні бази даних з одного сервера на інший.
- База даних по замовчуванню - master.
- Параметр **MUST_CHANGE** вказує на те, що при першому використанні нового імені входу, SQL Server буде здійснювати запит нового пароля. Але тут існує один нюанс: вікно для зміни пароля з'являється лише в SQL Server Management Studio. Більшість інших додатків при підключенні такого користувача повернуть помилку.
- Параметр **CREDENTIAL** на сьогоднішній день лише зв'язує з новим іменем входу об'єкт Credential, який являє собою пару «ім'я облікового запису Windows – пароль». Цей об'єкт використовується здебільшого у випадку, коли користувач хоче здійснити певні дії в ОС або на іншому сервері SQL Server з сценарію T-SQL. Об'єкт Credential можна створити за допомогою оператора CREATE CREDENTIAL.
- Параметр **CHECK_POLICY** буде працювати лише, якщо SQL Server працює під управлінням Windows 2003 Server або пізнішими версіями. Якщо він встановлений, то на пароль для даного логіна будуть просто поширюватись ті вимоги, які застосовуються до локальних облікових записів на даному комп'ютері.

Приведемо приклад створення логіну SQL Server з паролем.

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

Новий обліковий запис буде розміщений в директорії **Logins** (логіни, імена входу) папки **Security** (безпека) сервера.



А тепер створимо ім'я входу Windows, яке буде зв'язане з обліковим записом користувача Windows Pupkin з групи students, яка знаходиться в домені stepdom:

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

Якщо необхідно зареєструвати всіх користувачів з групи students домена stepdom, крім користувача Petrenko, слід використати оператор **DENY** (більш детальніше він буде розглянутий нижче).

```
create login [stepdom\students] from windows;     -- реєструємо групу користувачів
deny connect sql to [stepdom\students\Petrenko]; -- виключаємо з неї користувача
                                                  -- Petrenko
```

Список всіх імен входу розміщується в системному представленні **sys.sql_logins**. Крім того, для перегляду списку всіх імен входу на сервері можна також скористатись системною зберігаємою процедурою безпеки **sp_helplogins**:

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

Наприклад:

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

Results | Messages

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

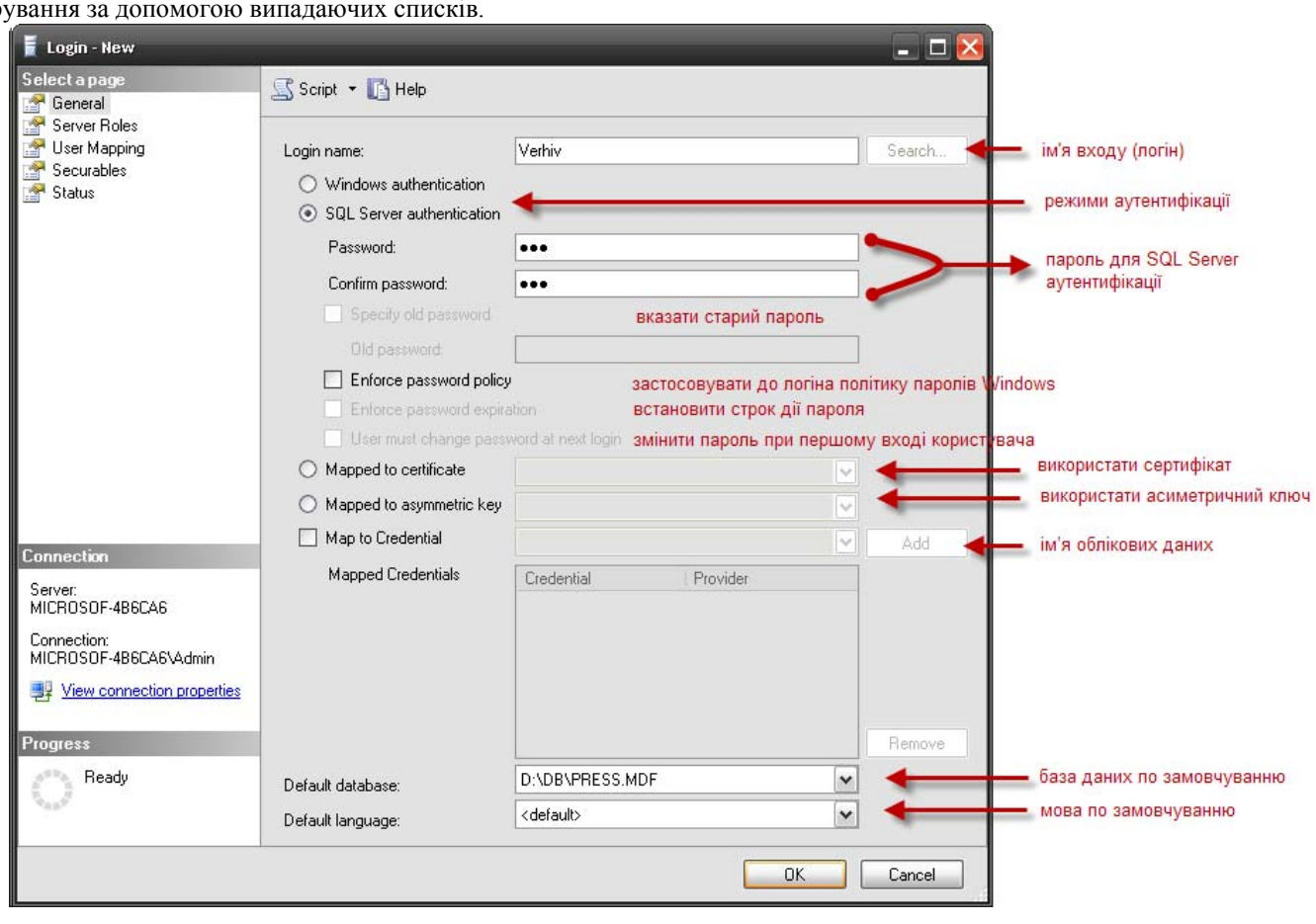
	LoginName	DBName	UserName	UserOrAlias
1	sa	D:\DB\PRESS.MDF	db_owner	MemberOf
2	sa	D:\DB\PRESS.MDF	dbo	User
3	sa	master	db_owner	MemberOf
4	sa	master	dbo	User
5	sa	model	db_owner	MemberOf
6	sa	model	dbo	User
7	sa	msdb	db_owner	MemberOf
8	sa	msdb	dbo	User
9	sa	tempdb	db_owner	MemberOf
10	sa	tempdb	dbo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

Результатом роботи даної системної зберігаємої процедури буде **2 звіти**:

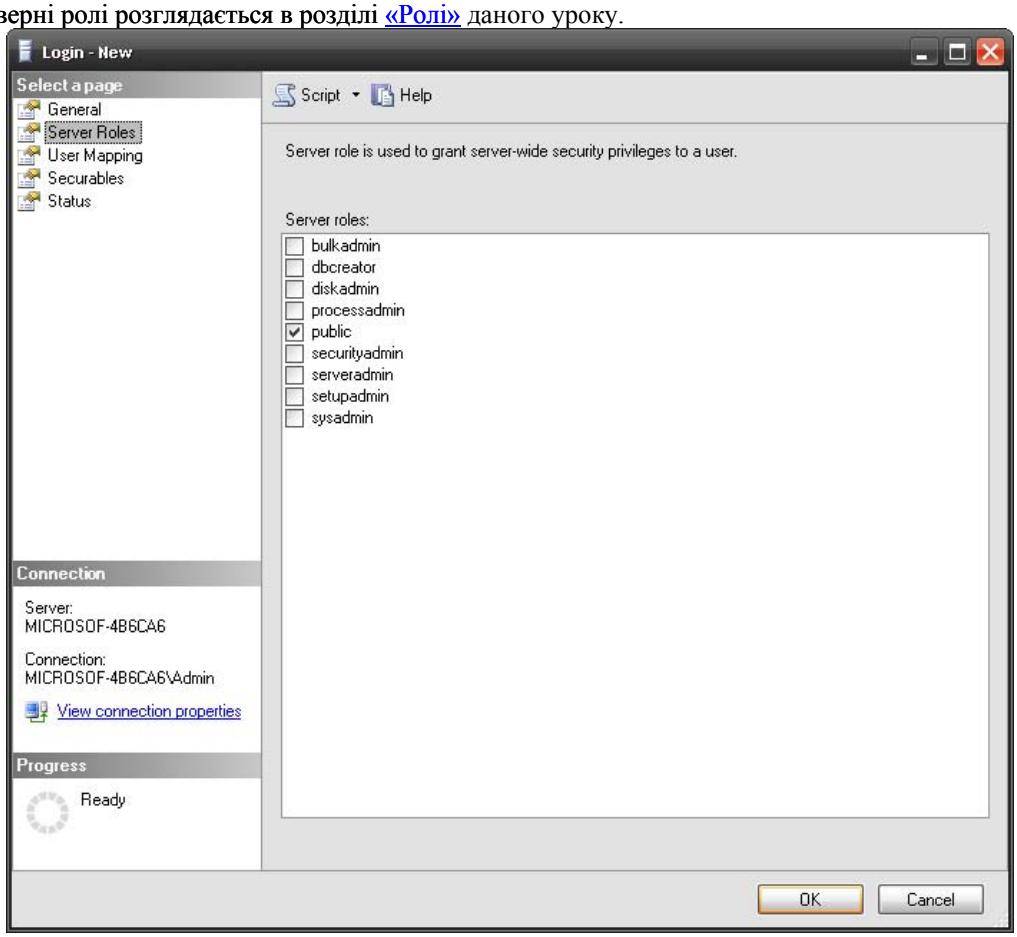
1. Перший звіт містить дані про ім'я входу.
2. Другий звіт містить узагальнені дані про користувачів, які відповідають даному імені входу та список ролей, членом яких він являється.

В SSMS існує папка **Security->Logins**, яка містить набір імен входу сервера баз даних. Щоб додати новий логін необхідно викликати пункт контекстного меню New->Login… гілки Security або New Login… дочірнього вузла Logins.

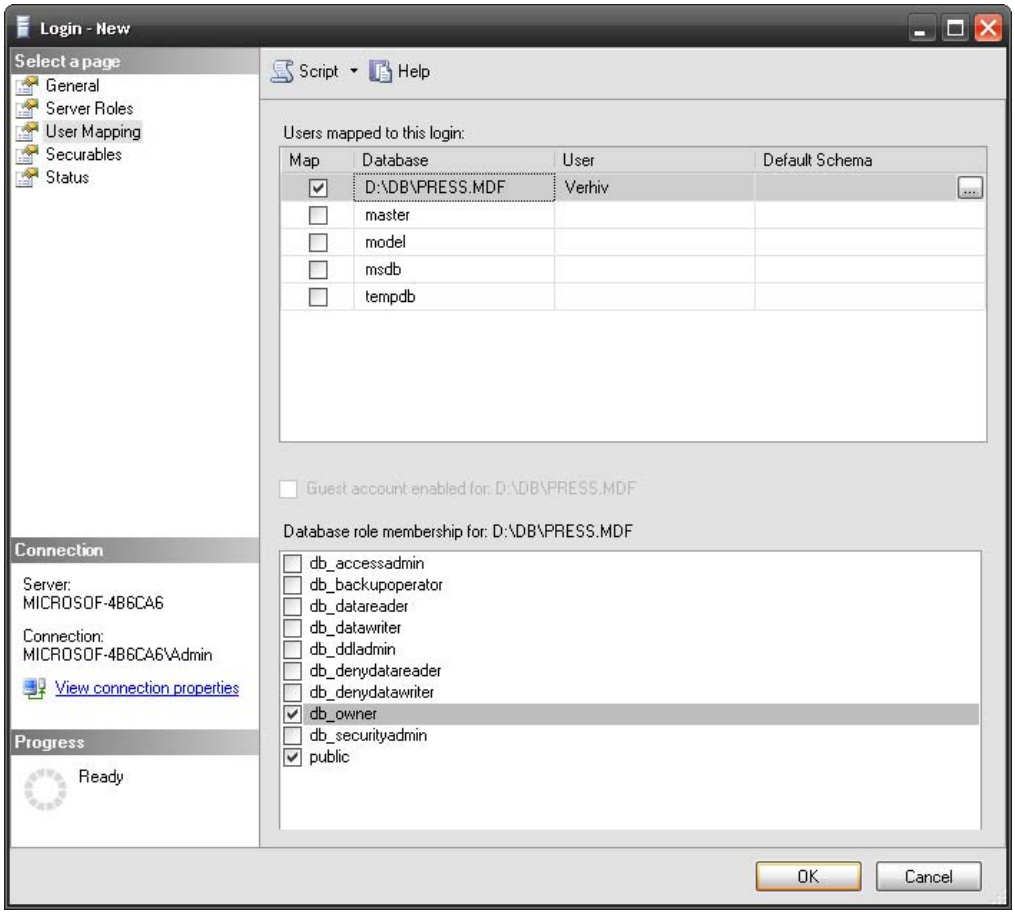
Після цього з'явиться діалогове вікно створення нового імені входу. На закладці **General (Загальні)** необхідно вказати загальні властивості для нового імені входу. В першу чергу необхідно задати саме ім'я входу та пароль для нього. Якщо ви створюєте ім'я входу Windows, тоді можна скористатись кнопкою Search (Пошук) для вибору існуючого облікового запису з Active Directory. У випадку створення імені входу з сертифікатом або асиметричним ключем слід обрати існуючий об'єкт шифрування за допомогою випадаючих списків.



Наступна закладка **Server Roles (Серверні ролі)** призначена для того, щоб вказати, яким фіксованим серверним ролям буде належати ім'я входу. По замовчуванню вона належить лише ролі PUBLIC. Тема ролей, включаючи і фіксовані серверні ролі розглядається в розділі «Ролі» даного уроку.



На закладці **співставлення користувачів** (**User Mapping**) оберіть базу даних Press, яку ми з вами створювали. Після цього необхідно назначити права на цю базу даних для новоствореного імені входу. Для прикладу, надамо йому повноцінні права на базу даних Press, тобто назначимо його ролі db_owner. Про ролі рівня бази даних розповідається в розділі «Ролі».

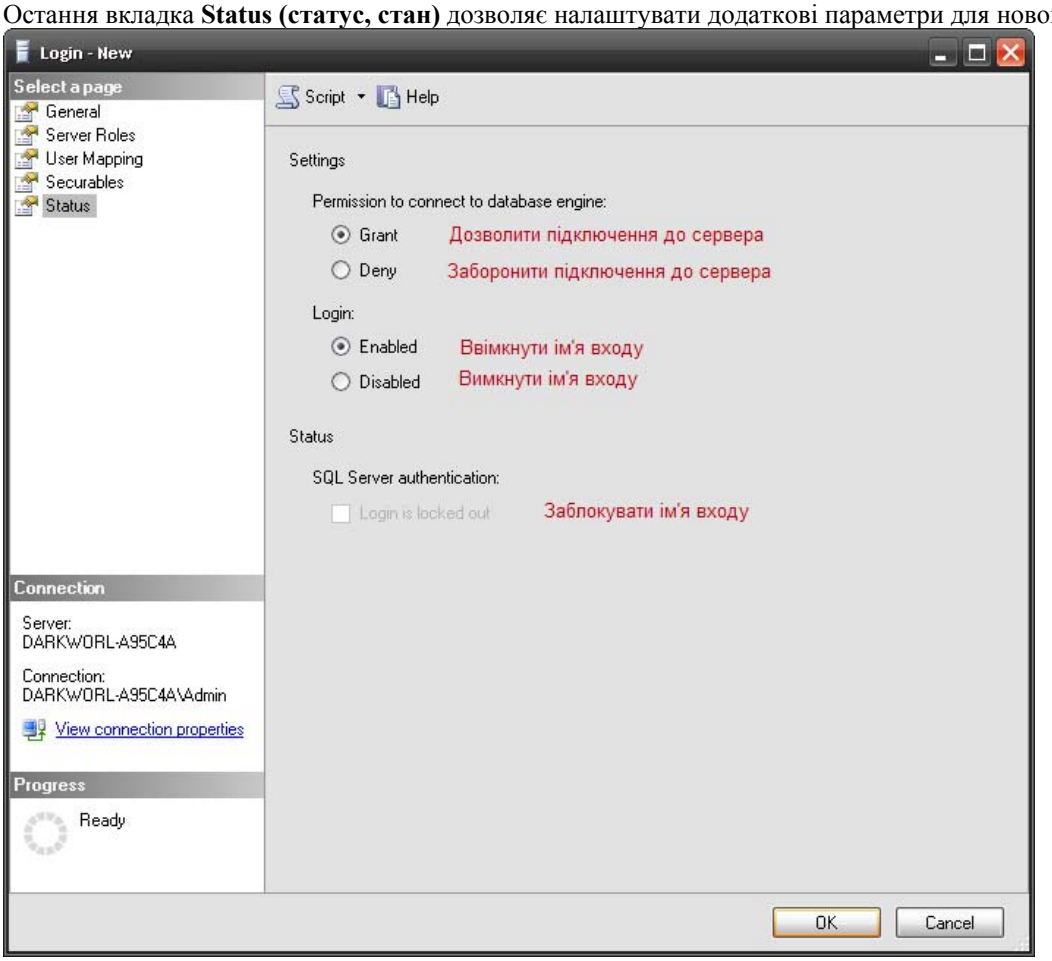


На вкладці **Securables (Захист)** ви маєте можливість одразу встановити права на:

- Обраний сервер та його об'єкти;
- Кінцеві точки HTTP (Endpoints);
- Інші імена входу (logins) та їх об'єкти.

Наприклад, ви можете одразу створити користувача, який зможе змінювати паролі для інших імен входу, які закріплені за працівниками його відділу.

Остання вкладка **Status (статус, стан)** дозволяє налаштувати додаткові параметри для нового імені входу.



Параметри тут наступні:

1. **Permissions to connect to database engine (дозвіл на підключення до ядра баз даних)** – значенням по замовчуванню для всіх логінів являється Grant, що вказує на дозвіл підключення до SQL Server. Значення Deny встановлюється, як правило, лише у випадку роздачі прав іменам входу групи Windows, але при цьому одному або кільком членам цієї групи планується заборонити доступ до сервера.
2. **Login (Ім'я входу)**, який визначає стан імені входу після створення: ввімкнений чи ні. Зазвичай всі логіни ввімкнені, але підстави для відключення також існують. Наприклад, звільнення працівника або переведення на його іншу посаду тощо. Найшвидшим способом вирішити дану стуацію буде просте відключення логіна.
3. **Status->SQL Server authentication->Login is locked out (Заблокувати логін)**. Ви можете лише зняти флажок блокування імені входу, якщо воно заблоковане. Ім'я входу автоматично блокується після використання всіх спроб невірного вводу пароля для вказаного логіна, а також якщо таке блокування налаштоване на рівні операційної системи і для логіна встановлена опція CHECK_POLICY.

Після натиснення кнопки OK створене вами ім'я входу буде відображатись в папці **Security->Logins**, як і раніше.

В SQL Server 2008 існує **набір вбудованих логінів**:

- **NT AUTHORITY\NetworkService** – ім'я входу мереженої служби, яке використовується для підключення до SQL Server Reporting Services. Він автоматично має права на бази даних master, msdb і на бази даних і служби, які використовує Reporting Services. Для SQL Server Agent дане ім'я входу використовувати не рекомендується.
- **NT AUTHORITY\LocalService** – ім'я входу локальної служби, яке має обмежений набір прав доступу до ресурсів (на рівні групи «Users» (Користувачі)). Цей логін звертається до мережених ресурсів у формі нулевого сеансу без

облікових даних. Логін локальної служби доступний лише в Microsoft Windows XP і Microsoft Windows Server 2003, а також не підтримується SQL Server Agent.

- **NT AUTHORITY\System** – локальний системний обліковий запис операційної системи. Він з'являється, якщо при інсталяції налаштувати роботу служби SQL Server від імені локального системного облікового запису. Він являється членом групи адміністраторів Windows на локальному комп'ютері та членом фіксованої серверної ролі sysadmin. Логін NT AUTHORITY\System існує лише для зворотної сумісності.
- **sa (System Administrator)** – ім'я входу, яке має повні і невід'ємні права системного адміністратора SQL Server. Це ім'я входу видалити не можна, його дозволяється лише перейменовувати або вимикати.
- Існує також **набір логінів, створених на основі сертифіката**, імена яких обмежені подвійним символом «хеш» (##). Такі імена входу використовуються лише для внутрішнього системного використання і видаляти їх не рекомендується:
  - ##MS_SQLResourceSigningCertificate##
  - ##MS_SQLReplicationSigningCertificate##
  - ##MS_SQLAuthenticatorCertificate##
  - ##MS_AgentSigningCertificate##
  - ##MS_PolicyEventProcessingLogin##
  - ##MS_PolicySigningCertificate##
  - ##MS_PolicyTsqlExecutionLogin##

2.3. Модифікація та видалення імен входу

Для зміни властивостей імен входу використовується оператор **ALTER LOGIN**, синтаксис якого представлений нижче:

```
ALTER LOGIN логін
{
    { ENABLE | DISABLE }      -- дозволити або заборонити використання імені входу
    | WITH список_опцій

    -- додати (або видалити) до імені входу облікові дані
    -- постачальника EKM (Extensible Key Management - розширеного управління ключами)
    | { ADD CREDENTIAL ім'я_облікових_даних | DROP CREDENTIAL ім'я_облікових_даних }
}

-- Список опцій:
[ PASSWORD = { 'пароль' | 'хешований_пароль' HASHED }
             [ OLD_PASSWORD = 'старий_пароль'
               | опції_пароля [ MUST_CHANGE | UNLOCK /*розблокувати логін*/]
             ]
]
[, DEFAULT_DATABASE = БД_по_замовчуванню ]
[, DEFAULT_LANGUAGE = мова_по_замовчуванню ]
[, NAME = логін ]                  -- нове ім'я для логіна, яке необхідно перейменувати
[, CHECK_POLICY = { ON | OFF } ] -- чи застосовувати до логіна політику паролів
                                   -- Windows на комп'ютері, де виконується SQL Server
[, CHECK_EXPIRATION = { ON | OFF } ] -- чи необхідно встановлювати строк дії пароля
[, CREDENTIAL = ім'я_облікових_даних ]
[, NO CREDENTIAL  ] -- видаляє існуючі співставлення логіну та облікових даних сервера
```

Параметр **DISABLE** оператора ALTER LOGIN не можна використовувати для заборони доступу групи Windows. Для цього призначений оператор **DENY**, який буде розглянутий нижче.

Іноді параметр **ENABLE** плутають з параметром UNLOCK. Щоб не було плутанини, пояснимо детальніше. Параметр ENABLE дозволяє використання імені входу, яке тимчасово було заборонене адміністратором. А параметр UNLOCK розблоковує ім'я входу, доступ якого був заборонений сервером через використання всіх спроб невірного вводу пароля для вказаного логіна. При цьому розблоковувати ім'я входу можна з новим паролем, а можна залишити йому старий, тоді як за допомогою ENABLE змінити пароль неможна, оскільки обліковий запис був просто тимчасово неактивний.

Приведемо кілька прикладів:

```
-- змінимо пароль, а саме відмінимо його
alter login Vasja
with password = '';

-- перейменувати ім'я входу
alter login Vasja
with name = VPupkin;
```

```
-- розблокувати ім'я входу з новим паролем
alter login Vasja
with password='111' unlock;
```

Для видалення імені входу використовується оператор **DROP LOGIN**:

```
-- синтаксис
DROP LOGIN логін

-- наприклад, необхідно видалити ім'я входу Windows
drop login [stepdom\students\Petrenko];
```

Варто відмітити, що для створення та управління іменами входу можна використовувати системні зберігаємі процедури безпеки. Наприклад, sp_addlogin, sp_defaultdb, sp_droplogin, sp_grantlogin тощо. Але всі ці зберігаємі процедури вважаються застарівшими та існують в SQL Server 2008 лише для забезпечення зворотньої сумісності. В наступній версії ці зберігаємі процедури будуть вилучені.

3. Користувачі бази даних

Отже, ім'я входу для користувача створили. Але цього, нажаль, не достатньо для того, щоб доступитись до даних в SQL Server. Користувач повинен мати в необхідній базі даних ще й свого користувача баз даних (database user). Таке розділення імен входу (login) і користувачів баз даних (user) в SQL Server забезпечує велику гнучкість. Тепер користувач, маючи один логін, може мати кілька користувачів в різних базах даних і з різними правами доступу.

Таким чином користувач після аутентифікації на сервері проходить так звану **авторизацію користувача**. В ході авторизації SQL Server перевіряє чи відповідає зареєстроване ім'я входу (login) хоча б одному з існуючих користувачів бази даних (user), до якої планується доступ, і які права він на неї має.

Створити користувача бази даних можна за допомогою оператора T-SQL **CREATE USER**:

```
CREATE USER ім'я_користувача
[ { { FOR | FROM }
    { LOGIN ім'я_входу                      -- ім'я входу SQL Server, для якого
                                            -- створюється користувач бази даних
      | CERTIFICATE ім'я_сертифіката -- сертифікат, для якого створюється користувач БД
      | ASYMMETRIC KEY ім'я_асиметричного_ключа  -- асиметричний ключ, для якого
                                                 -- створюється користувач бази даних
    }
    | WITHOUT LOGIN                 -- створює користувача, який буде
                                    -- підключатись до бази даних як користувач guest
]
[ WITH DEFAULT_SCHEMA = назва_схеми ]-- схема по замовч. для користувача БД, інакше dbo
```

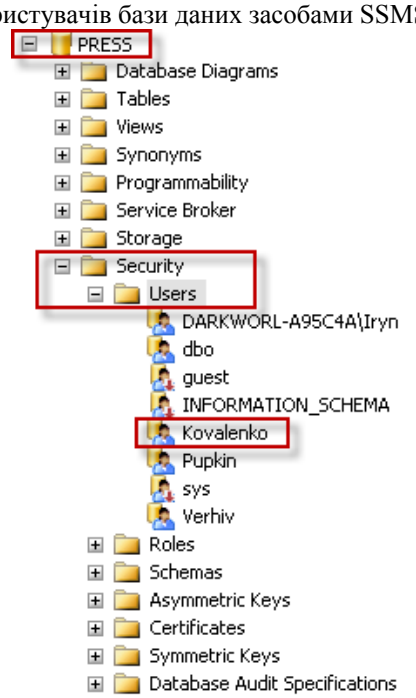
Зауваження щодо створення користувача:

- ✓ Якщо не вказати ім'я входу, тобто не задати аргумент **FOR LOGIN**, тоді новий користувач бази даних буде співставлятись з іменем входу SQL Server з аналогічним ім'ям.
- ✓ Значення **DEFAULT_SCHEMA** не можна вказувати для користувачів, які співставляються з групами Windows, сертифікатом або асиметричним ключем. Воно також ігнорується членами серверної ролі sysadmin, оскільки для них схемою по замовчуванню встановлена схема dbo.

Наприклад, створимо користувача бази даних для імені входу Kovalenko.

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

use Press;
create user Kovalenko
for login Kovalenko;
-- або
-- create user Kovalenko;
go
```

Нові користувачі бази даних додаються в групу **Users** вузла **Security** поточної бази даних. Отже, щоб створити користувачів бази даних засобами SSMS, слід викликати контекстне меню New User гілки Security або вузла Users.



У випадку доступу до бази даних імені входу, з яким не співставлений жоден користувач бази даних, SQL Server надасть доступ через користувача **guest**, якщо в нього є відповідні права. По замовчуванню користувач guest не має права доступу до баз даних, але ви можете йому їх надати, ввімкнувши його для необхідної бази даних:

```
-- ввімкнути користувача guest
grant connect to guest;
-- відмінити доступ користувача guest
revoke connect to guest;
```

Більш детальніше про оператори GRANT та REVOKE читайте у розділі «Управління правами доступу» даного уроку.

Щоб змінити властивості користувача, необхідно скористатись оператором **ALTER USER**, а щоб видалити користувача бази даних - **DROP USER**.

```
-- змінити властивості користувача баз даних
ALTER USER ім'я_користувача
WITH  [ NAME = нове_ім'я_користувача ]
      [, DEFAULT_SCHEMA = назва_схеми ]
      [, LOGIN = ім'я_входу ]

-- видалити користувача баз даних
DROP USER ім'я_користувача
```

Відмітимо, що для створення користувачів бази даних використовується також системна зберігаєма процедура **sp_grantdbaccess**, але вона вважається застарівшою і в наступній версії SQL Server буде вилучена, тому використовувати її не варто. Дані про всіх користувачів бази даних зберігаються в системному представленні **sys.server_principals**.

На рівні бази даних виділяють також поняття **«осиротівших» користувачів (orphaned users)**, які являють собою користувачів бази даних, з якими не співставляється жодне ім'я входу. В цю категорію попадають користувачі при видалені зв'язаних з ними імен входу. Для отримання інформації про таких користувачів використовується системна зберігаємо процедура **sp_change_users_login.**

```
-- синтаксис зберігаємої процедури sp_change_users_login
sp_change_users_login  [ @Action = ] 'дія'
                       [ , [ @UserNamePattern = ] 'користувач' ]
                       [ , [ @LoginName = ] 'імя_входу' ]
                       [ , [ @Password = ] 'пароль' ]
```

В якості **дії** можна вказувати одне з **наступних значень**:

- **Auto_Fix** – зв'язує запис користувача в системному представленні sys.database_principals поточної бази даних з іменем входу SQL Server, який має таке ж ім'я. З точки зору безпеки, намагайтесь уникати інструкції Auto_Fix, оскільки при відсутності необхідного імені входу, воно створюється.

  При використанні Auto_Fix необхідно дотримуватись наступних правил:
  - якщо імені входу ще не існує, тоді необхідно вказати аргументи user і password;
  - якщо ім'я входу вже присутнє, тоді слід вказати аргумент user без password;
  - аргумент login повинен мати значення NULL.

- **Report** – виводить список «осиротівших» користувачів і їх ідентифікатори безпеки (SID) в поточній базі даних. При цьому аргументи user, login і password повинні мати значення NULL або бути відсутніми.
- **Update_One** – зв'язує вказаний аргумент UserNamePattern в поточній базі даних з існуючим аргументом LoginName. Аргумент password повинен бути відсутній або рівний NULL.

Наприклад:

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

Результат:

Results | Messages

	UserName	UserSID
1	Pupkin	0xB0B3D59AF6FDF44CBD14F7005CCEEB26

При створенні нової бази даних, в ній автоматично створюються **4 користувачі**:

- **dbo (database owner)** – власник бази даних, яким автоматично стає той логін, від імені якого була створена база даних. Він входить до фіксованої ролі бази даних db_owner і має всі права на базу даних.
- **guest (гість)** – користувач, який надає право всім логінам, яким не відповідає жоден користувач бази даних.
- **INFORMATION_SCHEMA** – власник схеми INFORMATION_SCHEMA, в якій зберігаються представлення з системною інформацією для бази даних. Даному користувачу не відповідає жодне ім'я входу.
- **sys** - власник схеми SYS, до якої належать системні об'єкти бази даних. Даному користувачу не відповідає жодне ім'я входу.

4. Ролі

4.1. Поняття ролей. Фіксовані ролі

А тепер час розглянути ще одних учасників системи безпеки сервера – **ролі**, які дозволяють визначити рівень доступу (права) до об'єктів сервера баз даних. По суті, **роль** - це група користувачів, які мають однакові права. Наприклад, якщо в нас існує група користувачів, яким потрібен доступ лише для читання, то ми можемо створити роль READER, присвоїти їй необхідні права, а потім назначати цю роль конкретному користувачу. Тобто, якщо ролі надаються які-небудь права, то ці ж права має кожен користувач, який входить в дану роль.

Крім того, в SQL Server 2008 користувач може входити до складу кількох ролей. Це дозволяє групувати права доступу в певні ролі, а потім використовувати їх комбінації, створюючи тим самим набори прав, які найбільше підходять конкретному користувачу. В зв'язку з цим ролі сервера часто порівнюють з групами користувачів Windows.

Відмітимо, в кожній базі даних, включаючи системні бази даних такі як master, model та інші, існує роль **PUBLIC**. Це спеціальна роль, яка призначена для надання прав одразу всім користувачам бази даних. Вона має налаштовані по замовчуванню права для користувачів бази даних і ні її члени, ні вона сама не може бути знищена. Її членом автоматично стає кожен користувач бази даних.

Всі інші ролі можна розділити на наступні **<u>види</u>**:

1. **Ролі рівня сервера** (майже всі імена закінчуються на xxxadmin) – це ролі, які діють в рамках всього сервера і не належать жодній базі даних. Вони призначені для супроводження системи, включаючи управління обліковими записами та роботу з зв'язаними серверами. До вбудованих (фіксованих) серверних ролей відносяться:
   - **sysadmin** – користувачі даної ролі можуть виконувати будь-які дії з SQL Server, оскільки ця роль об'єднує в собі права всіх інших серверних ролей.
   - **serveradmin** – користувачі даної ролі можуть виконувати дії по конфігуруванню сервера: змінювати параметри сервера, завершувати його роботу тощо.
   - **setupadmin** – користувачі ролі можуть управляти (додавати, налаштовувати та видаляти) зв'язаними серверами, а також виконувати ряд системних зберігаємих процедур. Їм дозволяється також оголошувати процедури запуску.
   - **securityadmin** – дозволяє управляти іменами входу сервера, змінювати параметри налаштувань безпеки, включаючи віддалені сервери та права на створення бази даних, а також надає доступ до журналу помилок.
   - **processadmin** – дозволяє управляти процесами, які виконуються в екземплярі SQL Server.
   - **diskadmin** – користувачі даної ролі можуть управляти дисковими файлами (приєднувати і від'єднувати бази даних, вказувати, які файли входять до складу файлових груп тощо).

- **bulkadmin** – користувачі цієї ролі мають право на виконання оператора BULK INSERT, який використовується для масової вставки даних. При цьому дана роль не надає права доступу до таблиць, до яких буде застосований даний оператор.
- **dbcreator** – дозволяє створювати та управляти базами даних, включаючи відновлення з резервної копії.

Для того, щоб додати нового користувача або групу в серверну роль використовується системна зберігаєма процедура **sp_addsrvrolemember**, а для виключення одного з членів - **sp_dropsrvrolemember**.

```
-- додати до ролі
sp_addsrvrolemember [ @loginame= ] 'логін', [ @rolename = ] 'серверна_роль'

-- виключити з ролі
sp_dropsrvrolemember [ @loginame = ] 'логін', [ @rolename = ] 'серверна_роль'
```

Наприклад:

```
-- додамо до ролі sysadmin користувача Verhiv
exec sp_addsrvrolemember @loginame = 'Verhiv', @rolename = 'sysadmin'

-- виключити користувача Verhiv з серверної ролі diskadmin
exec sp_dropsrvrolemember 'Verhiv', 'diskadmin'
```

Відомості про членів серверної ролі можна отримати з системного представлення **sys.server_role_member**.

2. **Ролі рівня бази даних** (їх імена мають префікс db_xxx) існують на рівні бази даних та призначені для визначення прав по роботі з конкретною базою даних. До вбудованих ролей рівня бази даних відносять:
   - db_owner – дана роль назначається власникам бази даних, які мають на неї повні права.
   - db_accessadmin – користувачі даної ролі мають право роздавати і забирати права доступу до бази даних.
   - db_backupoperator - набір прав для резервного копіювання бази даних.
   - db_datareader – дана роль об'єднує всі права по зчитуванню даних з таблиць, представлень і функцій.
   - db_datawriter – користувачі даної ролі мають права на додавання, обновлення і видалення даних в поточній базі даних.
   - db_ddladmin – користувачі даної ролі мають права на управління даними в базі даних (INSERT, DELETE, UPDATE).
   - db_denydatareader – забороняє всім користувачам зчитувати дані бази даних (SELECT).
   - db_denydatawriter – забороняє всім користувачам управляти даними таблиць і представлень поточної бази даних (INSERT, DELETE, UPDATE).
   - db_securityadmin – дана роль об'єднує в собі всі права по адмініструванню системи захисту бази даних, тобто дозволяє управляти членами ролей та їх правами.

В SQL Server існує також можливість створювати власні ролі рівня бази даних за допомогою оператора CREATE ROLE. Більш детальніше про цю можливість читайте в п.4.2. «Користувацькі ролі» поточного розділу.

Щоб додати користувача, ім'я входу або іншу роль до ролі поточної бази даних, необхідно викликати системну зберігаєму процедуру **sp_addrolemember**:

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

Наприклад, необхідно до ролі рівня бази даних додати ім'я входу Windows.

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

При цьому до ролі рівня бази даних не можна додавати інші фіксовані ролі рівня бази даних, фіксовані серверні ролі або користувача **dbo**.

Видалити користувача, ім'я входу або іншу роль з ролі рівня бази даних допомагає серверна зберігаєма процедура **sp_droprolemember**.

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

3. **Ролі рівня додатку** дозволяють адміністратору бази даних обмежувати доступ користувачів до даних через прикладний додаток. Ця роль не містить в собі користувачів, тому перед використанням її необхідно активізувати в поточному з'єднанні, ввівши вірний пароль.

Процес роботи ролі рівня додатку відбувається наступним чином:

1) На сервері баз даних створюється роль рівня додатку (CREATE APPLICATION ROLE) і їй назначається певний набір прав.
2) Спочатку користувач реєструється в самому прикладному додатку передбаченим для цього способом, наприклад, за допомогою діалогового вікна.
3) Користувач перевіряється на існування і йому назначаються встановлені права.
4) Роль рівня додатку активізується (системна зберігаєма процедура sp_setapprole).
5) Перевіряється роль рівня додатку і з'єднання переключається в контекст цієї ролі. Варто відмітити, що при активізації ролі рівня додатку всі права ролей користувачів на сервері ігноруються і вони отримують лише права, які діють на рівні прикладного додатку. В зв'язку з цим, якщо користувачу необхідно знову повернутись до заданих для нього параметрів безпеки, він повинен закрити з'єднання з сервером і зареєструватись на сервері знову.

Отже, створити роль рівня додатку можна за допомогою оператора **CREATE APPLICATION ROLE**:

```
CREATE APPLICATION ROLE назва_ролі
WITH PASSWORD = 'пароль'
  [, DEFAULT_SCHEMA = назва_схеми ]
```

Аргумент DEFAULT_SCHEMA вказує на схему по замовчуванню, тобто на першу схему, в якій сервер буде шукати об'єкти для цієї ролі. Якщо вона не задана, то схемою по замовчуванню стає схема **dbo**. Доречі, вказана схема в базі даних може бути тимчасово відсутня.

Щодо паролей ролей рівня додатку, то вони повинні зберігатись в зашифрованому вигляді.

Після створення ролі рівня додатку, щоб нею скористатись її необхідно активізувати за допомогою зберігаємої процедури **sp_setapprole**:

```
sp_setapprole [ @rolename = ] 'назва_ролі'
              [, @password = ] { encrypt N'пароль_ролі' } | 'password'
              [, [ @encrypt = ] { 'none' | 'odbc' } ]
              [, [ @fCreateCookie = ] true | false ] -- чи потрібно створювати cookie файл
              [, [ @cookie = ] @cookie OUTPUT ]       -- вихідний параметр
                                                      --(тип varbinary(8000))
```

Аргумент **encrypt** дозволяє вказати тип шифрування пароля, але лише для з'єднань, які не використовують додаток SqlClient:

- none – кодування пароля не використовується, тобто пароль передається SQL Server у вигляді звичайного тексту (по замовчуванню);
- odbc – закодувати за допомогою функції ODBC encrypt перед відпракою компоненту SQL Server Database Engine. Але для захисту паролів, які передаються по мережі, її використовувати не слід, оскільки вона їх не шифрує. Слід відмітити, що даний аргумент може бути вказаний лише для клієнтів ODBC або постачальника даних OLE DB для SQL Server.

При використанні функції encrypt пароль повинен бути перетворений в рядок Unicode за допомогою модифікатора N. Якщо облікові дані необхідно зберігати в клієнтському додатку, тоді їх краще зашифровувати за допомогою функцій шифрування API.

Після ввімкнення ролі рівня додатку вона залишається активною до тих пір, поки користувач не відключиться від сервера або не виконає зберігаємо процедуру **sp_unsetapprole**.

Приведемо невеличкий приклад створення ролі додатку та її активації:

```
-- створюємо роль рівня додатку
create application role QRole
with password = 'AsDeF00MbXX';

-- активізуємо її
exec sp_setapprole 'QRole', 'AsDeF00MbXX';
```

Змінити властивості ролі додатку можна за допомогою оператора **ALTER APPLICATION ROLE**.

```
-- синтаксис
ALTER APPLICATION ROLE ім'я_ролі
WITH [ NAME = нове_ім'я_ролі ]
     [, PASSWORD = 'пароль' ]
     [, DEFAULT_SCHEMA = назва_схеми ]

-- наприклад, додамо до ролі додатку QRole схему по замовчуванню
alter application role QRole
with default_schema = sale;
```

Для видалення ролі рівня додатку використовують оператор **DROP APPLICATION ROLE**.

```
DROP APPLICATION ROLE ім'я_ролі
```

Застарілим аналогом оператора CREATE APPLICATION ROLE являється системна зберігаєма процедура **sp_addapprole**. Нагадаємо, що в SQL Server 2008 вона існує лише для сумісності з попередніми версіями.

Ролі рівня додатку існують лише на рівні бази даних. Тобто, якщо звернутися до іншої бази даних, коли прикладний додаток знаходиться в контексті безпеки ролі додатку, тоді доступ до неї буде здійснюватись з врахуванням прав облікового запису guest в цій базі даних. Якщо обліковий запис guest неактивний для цієї бази даних, доступ до неї буде заборонений.

Ролі додатку діють також під час сеансів. Отже, якщо ваш прикладний додаток працює з багатьма сеансами, які використовують одну роль, в такому разі кожен сеанс повинен її спочатку активізувати.

Ця роль дуже корисна у випадках, коли на рівні сервера необхідно обмежити доступ користувачів, передавши повноваження у розподілі прав певному прикладному додатку. Наприклад, тепер клієнтський прикладний додаток може для одних підключень використовувати контекст безпеки користувача, а для інших контекст безпеки ролі додатку.

При використанні ролі рівня додатку можуть бути корисними наступні **функції**:

- **USER_NAME([id])** - повертає ім'я поточного користувача або ім'я користувача з вказаним ідентифікатором;
- **SYSTEM_NAME** – повертає ім'я поточного імені входу.

4.2. Користувацькі ролі

Як вже було сказано раніше, SQL Server підтримує можливість створення власних ролей рівня бази даних, які будуть групувати користувачів баз даних з однаковими правами доступу до об'єктів. Це дозволить класифікувати користувачів по категоріям необхідності доступу.

Синтаксис оператора **CREATE ROLE**, який призначений для створення користувацької ролі бази даних має наступний вигляд:

```
CREATE ROLE ім'я_ролі [ AUTHORIZATION власник ]
```

По замовчуванню власником ролі є власник поточної бази даних, але його можна змінити на іншого користувача бази даних або зробити ним іншу роль. На практиці, даний аргумент при створенні ролі майже не використовується.

Застарівшим аналогом даного оператора є системна зберігаєма процедура sp_addrole.

Після створення в роль необхідно додати користувачів за допомогою системної зберігаємої процедури **sp_addrolemember**, а видалити користувача з ролі - **sp_droprolemember**.

Наприклад, створимо роль бази даних Management та додамо до неї користувача Verhiv.

```
create role Management;                          -- створюємо роль бази даних
exec sp_addrolemember 'Management', 'Verhiv'; -- додаємо в нього користувача
```

Для зміни властивостей ролі слід скористатись оператором **ALTER ROLE**. Але спектр її можливостей розповсюджується лише на зміну назви ролі.

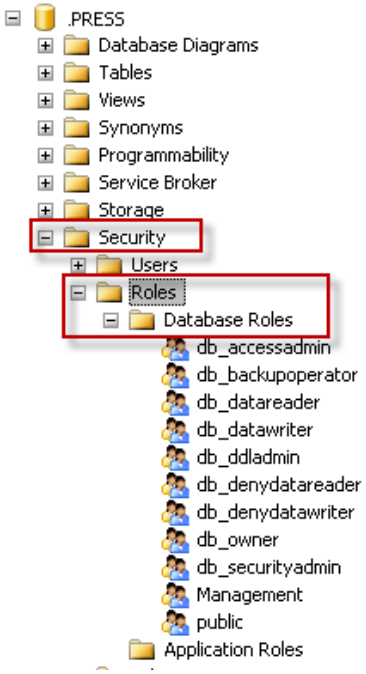
```
ALTER ROLE назва_ролі
WITH NAME = нове_ім'я_ролі
```

Для видалення користувацької ролі бази даних використовується оператор **DROP ROLE**:

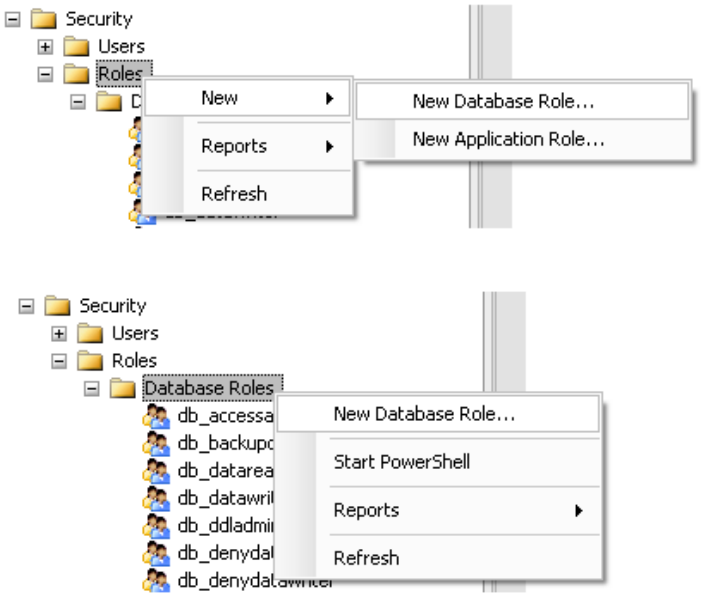
```
DROP ROLE назва_ролі
```

Створювати та управляти користувацькими ролями можна і засобами SSMS. Всі користувацькі ролі після створення розміщуються в гілці **Security->Roles** відповідної бази даних. Отже, і створювати їх можна за допомогою контекстного меню **New Role** вузла Roles або батьківського вузла Security.

Місцерозташування ролей



Додати нову роль



Щоб переглянути список ролей бази даних можна звернутись до системного представлення **sys.database_principals**, яка містить також перелік користувачів бази даних, або скористатись системною зберігаємою процедурою **sp_helprole**.

```
-- синтаксис системної зберігаємої процедури sp_helprole
sp_helprole [ [ @rolename = ] 'ім'я_ролі' ]

-- приклади:
-- переглянемо всі ролі поточної бази даних
exec sp_helprole;
-- інформація про членів ролі 'Management'
exec sp_helprole 'Management';
```

Результат:

Results | Messages

	RoleName	RoleId	IsAppRole
1	public	0	0
2	Management	9	0
3	db_owner	16384	0
4	db_accessadmin	16385	0
5	db_securityadmin	16386	0
6	db_ddladmin	16387	0
7	db_backupoperator	16389	0
8	db_datareader	16390	0
9	db_datawriter	16391	0
10	db_denydatareader	16392	0
11	db_denydatawriter	16393	0

← `exec sp_helprole;`

	RoleName	RoleId	IsAppRole
1	Management	9	0

← `exec sp_helprole 'Management';`

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

Результат:

Results | Messages

	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	N...	0
5	sys	4	S	SQL_USER	NULL	2005-10-14 ...	2005-10-14 ...	NULL	N...	0
6	DARKWORL-A95C4A\Iryn	5	U	WINDOWS_USER	dbo	2007-04-02 ...	2007-04-02 ...	NULL	0x...	0
7	Pupkin	6	S	SQL_USER	dbo	2007-11-03 ...	2007-11-03 ...	NULL	0x...	0
8	Verhiv	7	S	SQL_USER	dbo	2010-09-27 ...	2010-09-27 ...	NULL	0x...	0
9	Kovalenko	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	16384	R	DATABASE_ROLE	NULL	2003-04-08 ...	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

Розшифруємо коротко структуру даного системного представлення:

Поле	Опис
name	Учасник бази даних (роль або користувач).
principal_id	Ідентифікатор учасника бази даних.
type	Тип учасника: ▪ S - користувач SQL; ▪ U - користувач Windows; ▪ G - група Windows; ▪ A - роль рівня додатку; ▪ R - роль рівня бази даних; ▪ C – користувач, співставлений з сертифікатом; ▪ K - користувач, співставлений з асиметричним ключем.
type_desc	Опис типу учасника.
default_schema_name	Схема по замовчуванню. Рівна NULL для учасників, які не належать типу S, U або A.
create_date	Дата створення учасника.
modify_date	Час останньої модифікації учасника.
owning_principal_id	Ідентифікатор власника учасника. Власником всіх учасників, крім ролей баз даних, являється dbo.
sid	Ідентифікатор захисту (SID), якщо учасник належить типу S, U або G, інакше NULL.
is_fixed_role	Флаг, який вказує на вбудовану (фіксовану) роль.

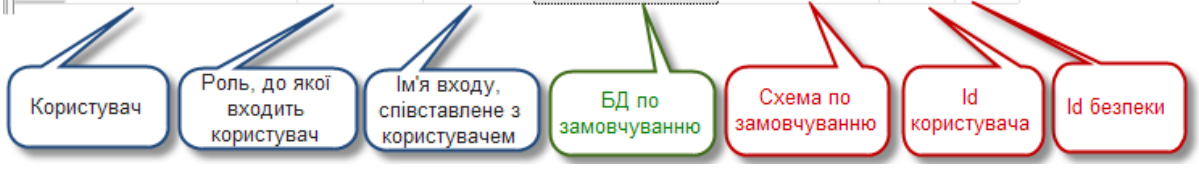
Для перегляду членів ролей бази даних, використовується системне представлення **sys.database_role_members** та зберігаєма процедура **sp_helpuser**. Але sp_helpuser не повертає інформацію про захищені об'єкти SQL Server 2008, тому для цих цілей краще скористатись системним представленням sys.database_principals.

```
exec sp_helpuser;
```

Результат:

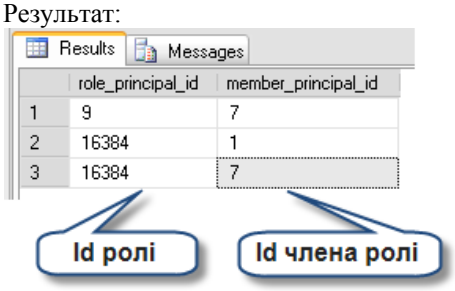
Results | Messages

	UserName	RoleName	LoginName	DefDBName	DefSchemaName	UserID	SID
1	DARKWORL-A95C4A\Iryn	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	Verhiv	Management	Verhiv	D:\DB\PRESS.MDF	dbo	7	0x...
9	Verhiv	db_owner	Verhiv	D:\DB\PRESS.MDF	dbo	7	0x...



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

Результат:



Як видно з результату, до ролі Management (role_principal_id = 9) поточної бази даних Press, входить один користувач з іменем Verhiv (member_principal_id = 9). Роль db_owner (role_principal_id = 16384) цієї ж бази даних включає в себе двох користувачів: Verhiv та dbo (member_principal_id = 1).

5. Управління правами доступу

5.1. Основні поняття

Після створення ролі або конкретного користувача та встановлення для них прав доступу до бази даних, їм необхідно надати права (привілеї) на об'єкти самої бази даних. Це необхідно для організації додаткового рівня безпеки даних. Наприклад, на підприємстві працівникам відділу постачання необхідно надати лише доступ до даних, які стосуються клієнтів, поставки і товару та заборонити переглядати інформацію про працівників підприємства, включаючи дані про зарплату. ☺ Здійснюється таке управління правами за допомогою операторів **GRANT**, **DENY** і **REVOKE**.

Права – це дозвіл користувачу здійснити певну операцію над об'єктом бази даних. Існують ряд видів об'єктів та операцій, на які можна встановлювати права для користувачів та ролей.

Права доступу, які отримує користувач, можна поділити на **наступні типи**:

1. **Неявні права** – це набір прав, які визначаються приналежністю до певної фіксованої ролі (сервера, бази даних або ролі рівня додатку) користувача;
2. **Об'єктні права** – це права по управлінню об'єктами бази даних;
3. **Командні права** – права на виконання операцій над базою даних.

5.2. Об'єктні права доступу

В SQL Server, починаючи з версії SQL Server 2005, об'єктів в базі даних, на які можна назначити права, стало більше. Тепер, крім роздачі прав на користування таблицями, представленнями, функціями і зберігаємими процедурами, можна назначати права на управління такими об'єктами, як ролі, користувачі бази даних, повнотекстові каталоги тощо. Таким чином, до захищених об'єктів відносяться всі ресурси, доступ до яких регулюється системою авторизації компонента SQL Server Database Engine. Крім того, ряд захищених об'єктів можуть зберігатись всередині інших, створюючи так звані **ієрархії областей**, для яких також можна встановлювати права доступу.

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

I. **Сервер**, який містить в собі такі захищаємі об'єкти, як:
- ім'я входу (login), тобто можна одному користувачу SQL Server надати дозвіл на об'єкт іншого користувача;
- база даних;
- кінцеві точки (endpoint) HTTP.

III. **База даних**, яка включає наступні захищаємі об'єкти:
- користувач (user);
- роль;
- роль рівня додатку;
- збірка (assembly) .NET;
- тип повідомлень, який дозволяє здати дозвіл на захищаємий об'єкт компонента Service Broker;
- маршрут;
- служба;
- прив'язка віддаленої служби;
- повнотекстовий каталог;
- сертифікат;
- симетричний і асиметричний ключ;
- контракт;
- схема.

III. **Схема**, яка містить такі захищаємі об'єкти:
- користувацький тип даних (домен);
- колекція схем XML;
- об'єкт:
  - таблиця;

- представлення;
- функція;
- зберігаєма процедура;
- обмеження;
- черга;
- синонім;
- статистика;
- статистичне обчислення.

Це нововведення дозволяє суттєво полегшити роботу адміністратору баз даних, оскільки надавши користувачу права на область, наприклад, на управління схемою, ви надаєте йому одразу права на користування всіма об'єктами цієї схеми.

Права на об'єкти бази роздаються за допомогою оператора **GRANT**. В ньому дуже багато опцій і повне знайомство з даним оператором виходить за межі нашого уроку. Ми ж спробуємо розглянути лише основні принципи його використання. Узагальнений синтаксис оператора GRANT виглядає наступним чином:

```
GRANT { ALL [ PRIVILEGES ] }
      | назва_привілею [ ( назва_поля [ ,...n ] ) ] [ ,...n ]
[ ON [ клас_захищаємого_об'єкта :: ] захищаємий_об'єкт ]
TO ім'я_облікового_запису [ ,...n ]
[ WITH GRANT OPTION ]
[ AS ім'я_облікового_запису ]
```

Ключове слово **ALL** вказує на те, що користувачу необхідно надати всі права на вказаний об'єкт. Але в SQL Server 2008 воно не надає всі права. Більше того, ключове слово ALL являється застарівшим і існує лише для сумісності з попередніми версіями. В наступній версії даний параметр планується виключити, тому використовувати його не варто.

Ключове слово **PRIVILEGES** також являється застарівшим і було введено в попередніх версіях лише для відповідності стандарту SQL-92.

Якщо ключове слово ALL не використовується, тоді вказується список прав доступу. Після ключового слова **ON** розповідаємо на який саме об'єкт роздаються права та кому (**TO**). Отримувачем прав може бути користувач бази даних, ім'я входу (включаючи ім'я входу або групу Windows) або роль.

Опція **WITH GRANT OPTION** вказує на те, що користувачі, яким надаються права на об'єкт (-и) бази даних, можуть надавати їх іншим користувачам.

Якщо обліковий запис належить кільком ролям, тоді необхідно при роздачі прав вказати опцію **AS**. Після цього ключового слова задається список ролей, які конкретизують користувача: користувачу з якої саме ролі надати ті чи інші права.

Як видно з синтаксису, в SQL Server можна встановлювати права на рівні окремих полів. Але на практиці краще не користуватись такими правами через втрату продуктивності і ускладнення системи прав. Для таких цілей, тобто заборони доступу користувача або ролі до певних полів таблиці чи іншого об'єкта, краще використати представлення або зберігаєму процедуру, які будуть показувати лише необхідну інформацію.

Для прикладу, розберемо права, які найчастіше роздаються, тобто права на користування об'єктами бази даних. Отже, в SQL Server можна надавати наступні **об'єктні права**:

1. **SELECT, INSERT, UPDATE, DELETE** на таблиці, представлення, табличні функції і на поля перелічених об'єктів, а також на синоніми.
2. Право на запуск зберігаємих процедур та скалярних функцій – **EXECUTE**.
3. Право **REFERENCES** двояке, оскільки його можна надавати таблицям, представленням, функціям, які повертають табличне значення, та чергам компонента Service Broker. Таким чином:
   - право REFERENCES для таблиці необхідне для створення обмеження на зовнішній ключ (FOREIGN KEY), який ссилається на цю таблицю;
   - право REFERENCES для представлення або табличної функції необхідне для сворення цих об'єктів з параметром WITH SCHEMABINDING.

4. **ALTER** – можливість змінювати властивості об'єкта (по замовчуванню це право має лише його власник). На рівні бази даних можна налаштовувати права **ALTER ANY об'єкт_сервера_або_БД**, які надають можливість вносити зміни в довільні об'єкти сервера або бази даних. Наприклад, ALTER ANY LOGIN надає право створювати, змінювати або видаляти довільне ім'я входу в екземплярі, а ALTER ANY TABLE надає можливість управляти таблицями бази даних.

   Слід також пам'ятати, що якщо ви встановите право ALTER на схему, це дозволить управляти всіма об'єтами, що входять до цієї схеми.

4. **CONTROL** – надає повні права на об'єкт, включаючи інформацію в ньому. Крім того, користувачу, який має право CONTROL, дозволяється роздавати права на встановлений об'єкт іншим. Право CONTROL на певну область неявно включає право CONTROL на всі об'єкти, які знаходяться в її межах. Наприклад, привілей CONTROL на базу даних

неявно надає всі права на базу даних, всі права на її збірки (assembly), на її схеми і об'єкти, які знаходяться в межах цих схем.

5. **TAKE OWNERSHIP** - право на передачу прав власності на об'єкт. Таке право можна назначити для будь-яких об'єктів.
6. **VIEW DEFINITION** - право на перегляд метаданих таблиць, представлень, процедур або функцій.
7. **VIEW CHANGE TRACKING** – право на відслідковування змін в таблиці (або схемі), наприклад, буде надана інформація про те, які поля були змінені. Повний список операцій, які відслідковуються наступний:
   - DROP TABLE;
   - ALTER TABLE DROP CONSTRAINT;
   - ALTER TABLE DROP COLUMN;
   - ALTER TABLE ADD COLUMN;
   - ALTER TABLE ALTER COLUMN;
   - ALTER TABLE SWITCH;
   - DROP INDEX або ALTER INDEX DISABLE;
   - TRUNCATE TABLE.

8. **RECEIVE** – право на роботу з чергами компонента Service Broker.

Щоб краще зрозуміти роботу оператора GRANT, розглянемо кілька **прикладів**:

1. Надати права на вибірку даних з таблиці book.Authors користувачу Verhiv:

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

2. Якщо необхідно надати права одразу кільком користувачам:

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

3. Надати ролі Management права на перегляд та зміну даних таблиці sale.Sales:

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

4. Якщо необхідно видати права на перегляд і вибірку даних таблиці sale.Shops всім користувачам поточної бази даних, тоді можна використати роль **PUBLIC**, яке еквівалентне переліку всіх користувачів.

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

5. Для обмеження прав на перегляд таблиці sale.Sales лише кількома полями, необхідно написати наступний запит.

```
grant select on sale.Sales to public;

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

В результаті всі користувачі можуть лише переглядати таблицю sale.Sales, а користувач Kovalenko має ще право змінювати в ній поля Price та Quantity. Крім того, він має право передавати ці привілеї іншим користувачам.

6. Надамо право на запуск зберігаємої процедури sp_BestAuthors ролі Secretary:

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

Оператор **DENY** дозволяє встановити заборону на використання об'єктів бази даних. Особливістю його використання є те, що він являється вищим по пріоритету за оператор GRANT, тому дозволяє відмінити права користувача, видані раніше. Але заборона (DENY) на рівні таблиці має меньший пріоритет, ніж дозвіл рівня поля, виданий оператором GRANT (!). Ця несумісність прав в SQL Server 2008 існує лише для забезпечення зворотної сумісності і в наступній версії вона буде усунена.

Повний синтаксис оператора DENY також доволі складний і дещо нагадує синтаксис оператора GRANT. Його узагальнений спрощений вигляд представлений нижче:

```
DENY { ALL [ PRIVILEGES ] }
      | назва_привілею [ ( назва_поля [ ,...n ] ) ] [ ,...n ]
[ ON [ клас_захищаємого_об'єкта :: ] захищаємий_об'єкт ]
TO ім'я_облікового_запису [ ,...n ]
[ CASCADE ]
[ AS ім'я_облікового_запису ]
```

Як бачите, майже всі параметри аналогічні оператору роздачі прав. Новим параметром є лише **CASCADE**, який виконує зворотну дію по відношенню до опції WITH GRANT OPTION інструкції GRANT. Він дозволяє заборонити права вказаному користувачу і всім іншим користувачам, яким вони були надані за допомогою опції WITH GRANT OPTION. При цьому, якщо забрати права у користувача, якому цей привілей був наданий за допомогою опції WITH GRANT OPTION, і не вказати аргумент CASCADE, SQL Server згенерує помилку.

Для прикладу, надамо всім користувачам, крім користувача Verhiv право переглядати інформацію про магазини, які реалізують книги видавництва:

```
grant select on book.Books to public;     -- надаємо право select всім користувачам
deny select on book.Books to Verhiv;      -- відміняємо право select в користувача Verhiv
```

Щоб забрати надані права у користувачів, використовується оператор **REVOKE**. По суті, даний оператор дозволяє відмінити дії, здійснені раніше оператором GRANT або DENY.

Формат даної команди схожий з форматом команди GRANT.

```
REVOKE [ GRANT OPTION FOR ]
       { ALL [ PRIVILEGES ]
         | назва_привілею [ ( назва_поля [ ,...n ] ) ] [ ,...n ]
       }
[ ON [ клас_захищаємого_об'єкта :: ] захищаємий_об'єкт ]
{ TO | FROM } ім'я_облікового_запису [ ,...n ]
[ CASCADE ]
[ AS ім'я_облікового_запису ]
```

Опція **GRANT OPTION FOR** дозволяє забрати права на роздачу прав користувачем. При використанні параметра CASCADE дану опцію обов'язково слід вмикати.

Параметр **CASCADE** вказує на каскадну відміну прав, які надавались за допомогою параметра WITH GRANT OPTION оператора GRANT.

Наприклад, щоб забрати права на читання з таблиці sale.Sales у користувача Kovalenko потрібно написати наступне:

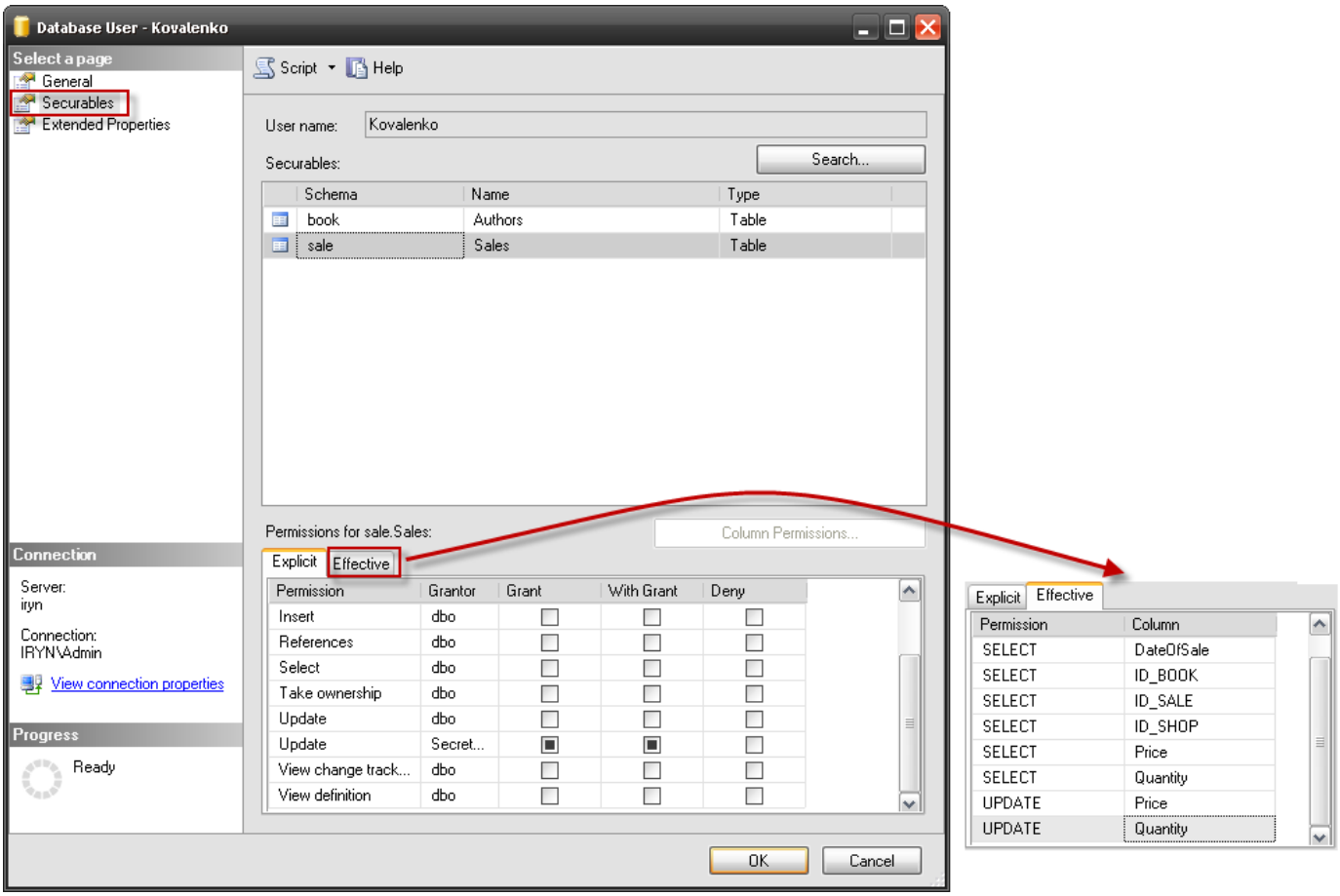
```
revoke select
on sale.Sales
from Kovalenko;
```

Якщо необхідно забрати права в кількох користувачів, тоді їх слід перелічити через кому. А для анулювання прав у всіх користувачів вкажіть роль PUBLIC.

В наступному прикладі заберемо право на роздачу привілея select на поля Price та Quantity таблиці sale.Sale в користувача Kovalenko.

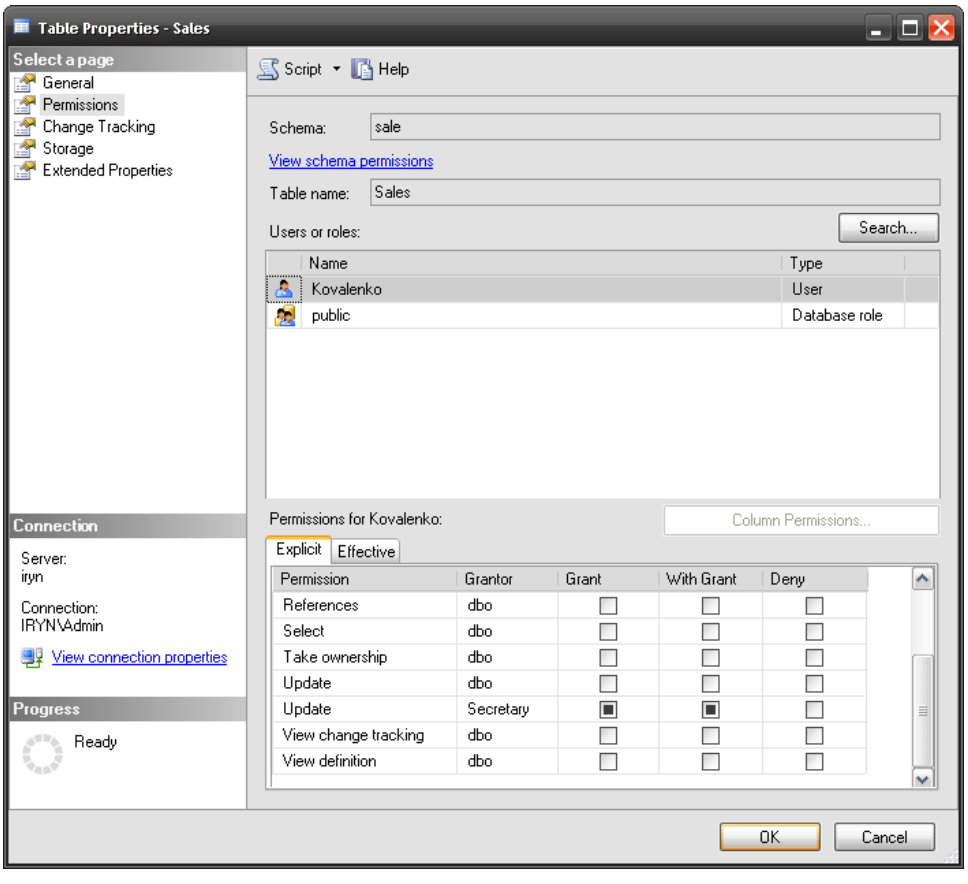
```
revoke grant option for update
on sale.Sales(Price, Quantity)
from Kovalenko;
```

Для надання або видалення прав SSMS має також свій власний, доволі зручний у використанні, набір засобів. Наприклад, для того, щоб переглянути і змінити набір прав певного користувача або ролі бази даних, необхідно перейти до їх **властивостей (Properties)** та обрати вкладку **Securables (Захист)**. Аналогічна вкладка існує при створенні окремо взятого користувача (або ролі бази даних). Виглядає вона так:



Даний інструмент дозволяє переглянути, на які об'єкти бази даних має права користувач. У випадку необхідності, новий об'єкт можна додати за допомогою кнопки **Search… (Пошук)**. В частині **Permissions for (Права доступу для)** детальніше представлено, які саме права має користувач на обраний об'єкт бази даних. На вкладці **Explicit** можна переглянути та налаштувати **явні** права для користувача. А на вкладці **Effective** – підсумкові права доступу для користувача або ролі бази даних (оскільки права від різних ролей бази даних, назначених цьому користувачу, сумуються).

Отримати інформацію про те, хто і які права має на окремо взятий об'єкт, можна за допомогою вкладки **Permissions** властивостей цього об'єкта. Наприклад, для таблиць вона матиме наступний вигляд:



Список **Users or roles (Користувачі або ролі)** дозволяє легко переходити між користувачами або ролями бази даних, які мають права на цю таблицю. В частині **Permissions for (Права доступу для)** ви також можете більш детальніше ознайомитись з набором прав поточного (виділеного) користувача або ролі бази даних.

Насамкінець відмітимо, що на практиці використовується більше десятка таблиць і інших об'єктів бази даних, доступ до яких надавати кожному користувачу доволі незручно. Спростити цю задачу допомагає встановлення прав доступу на рівні схеми або всієї бази даних, а також використання фіксованих ролей рівня бази даних.

Також не забувайте про те, що доступ користувачів на пряму до таблиць організовувати не варто. Натомість зміну даних краще організувати за допомогою зберігаємих процедур, а для перегляду даних використати представлення або ті ж зберігаємі процедури.

5.3. Командні права доступу

В SQL Server користувач, крім об'єктних прав, може мати також набір **командних прав**, які дозволяють йому виконувати певний набір операторів (команд). Фактично – це набір прав на базу даних. В SQL Server 2008 **список командних прав** доволі великий і має наступний вигляд:

- 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.

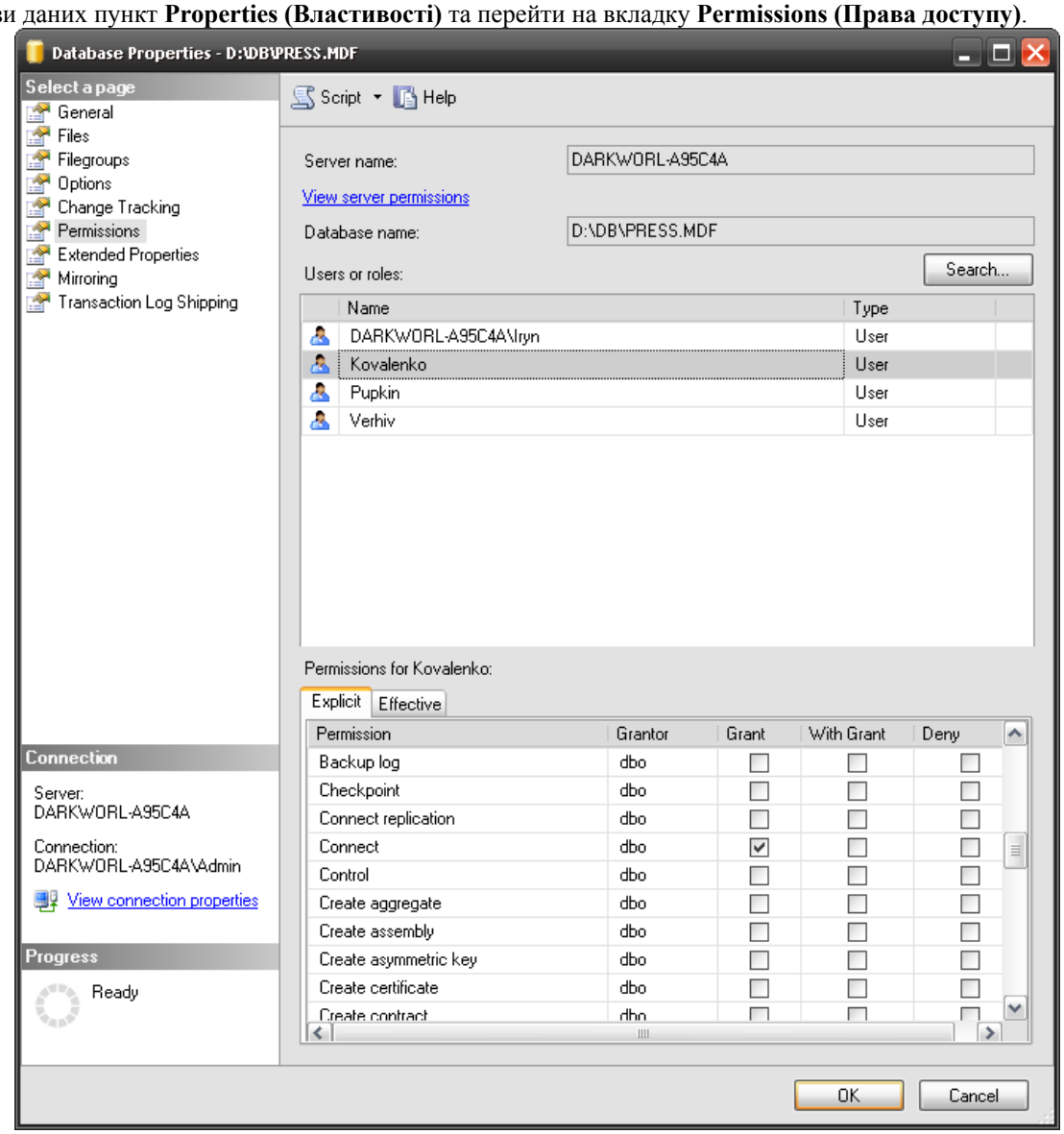
Управління командними правами нічим особливим від управління об'єктними не відрізняється. Для цього також використовуються оператори GRANT, DENY та REVOKE, але дещо в іншій формі.

```
-- роздача прав
GRANT  { ALL [ PRIVILEGES ] | назва_привілею [ ,...n ] }
TO ім'я_облікового_запису [ ,...n ]
[ WITH GRANT OPTION ]
[ AS ім'я_облікового_запису ]

-- анулювати або забрати права
{ DENY | REVOKE}  { ALL  [ PRIVILEGES ] | назва_привілею [ ,...n ] }
TO ім'я_облікового_запису [ ,...n ]
[ CASCADE ]
[ AS ім'я_облікового_запису ]
```

Приведемо приклади управління такими правами:

```
-- надамо права користувачу Kovalenko на створення таблиці
grant create table
to Kovalenko;

-- заберемо в ролі Management права на створення представлення
revoke create view
to Management;
```

Щоб налаштувати командні права доступу до бази даних за допомогою SSMS, необхідно обрати в контекстному меню бази даних пункт **Properties (Властивості)** та перейти на вкладку **Permissions (Права доступу)**.



Цей графічний інтерфейс дозволяє роздавати або анулювати обране право для поточного користувача чи ролі бази даних, шляхом простого ростановляння або знімання галочок в частині **Permissions for (Права доступу для)**.

5.4. Дані про права

Для отримання інформації про те, хто і які права має, крім вже розглянутих способів (з властивостей окремо взятого об'єкта), можна скористатись ще системною зберігаємою процедурою **sp_helprotect**. Вона дозволяє переглянути інформацію про права доступу до об'єктів поточної бази даних.

```
sp_helprotect [ [ @name = ] 'оператор_або_об'єкт_БД' ] -- інформацію про які права
                                                       -- відобразити
     [ , [ @username = ] 'ім'я_облікового_запису' ]    -- права якого учасника перевіряються
     [ , [ @grantorname = ] 'ім'я_облікового_запису' ]-- яким користувачем були надані права
     [ , [ @permissionarea = ] 'тип' ]                 -- тип прав, які відображаються
```

Як видно з синтаксису, аргументи зберігаємої процедури являються необов'язковими. Таким чином, якщо її запустити на виконання без аргументів, на екран буде виведений список всіх прав, які були надані або заборонені для поточної базі даних.

Зауваження щодо аргументів зберігаємої процедури:

1. Якщо аргумент **name** являється оператором, тоді він може приймати лише одне з наступних значень:
   - CREATE DATABASE;
   - CREATE DEFAULT;
   - CREATE FUNCTION;

- CREATE PROCEDURE;
- CREATE RULE;
- CREATE TABLE;
- CREATE VIEW;
- BACKUP DATABASE;
- BACKUP LOG.

2. Аргумент **permissionarea** вказує на те, які права будуть відображені: на об'єкти (символ «o»), оператори (символ «s») або на те і інше («os»; значення по замовчуванню). При використанні комбінацій типів, символи «o» і «s» розділяються комою або пропуском.
3. Якщо необхідно вказати лише кілька аргументів, тоді замість пропущених слід вказувати значення NULL або використати іменовані аргументи. Наприклад, необхідно переглянути всі права, які має користувач Kovalenko:

```
exec sp_helprotect NULL, Kovalenko
-- або
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	.

Дані в результуючій множині сортуються по категорії прав, власнику, об'єкту, отримувачу прав, участнику, який надав привілей, категорії типу захисту, типу захисту, дії і по полю ідентифікатора.

Приведемо ще один приклад використання зберігаємої процедури sp_helprotect. Виведемо список прав на таблицю 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)

Оскільки розглянута зберігаєма процедура не повертає інформацію про захищаємі об'єкти, які з'явились ще в SQL Server 2005, для таких цілей рекомендовано використовувати представлення **sys.database_permissions** та функцію **fn_builtin_permissions**. Більш детальніше про них читайте в документації по SQL Server 2008.

6. Резервне копіювання БД та відновлення з резервної копії

6.1. Основні поняття

От і підійшли ми до завершення курсу «SQL Server 2008». Створювати базу даних ми навчились, а також розібрались з організацією безпеки даних на сервері. Але це ще далеко не все. Після цього варто потурбуватись про оптимізацію її діяльності та безпеку даних у випадку виникнення непередбачуваних ситуацій, адже в процесі роботи всяке трапляється: пожежа, потоп, землетрус, різного роду збої в апаратній частині, помилки в роботі користувачів (наприклад, були випадково видалені або змінені дані) тощо. Для таких випадків дуже корисно мати під рукою копію даних, яку можна відновити у випадку втрати інформації. Здійснюється це за допомогою **механізму резервного копіювання**.

Резервне копіювання (backup) бази даних та **відновлення з резервної копії** (**restore**) – дві важливих та найбільш частих адміністративних процеси, які здійснюються розробниками і адміністраторами бази даних. В SQL Server 2008 для цього можна скористатись операторами T-SQL або використати набір інструментів SSMS.

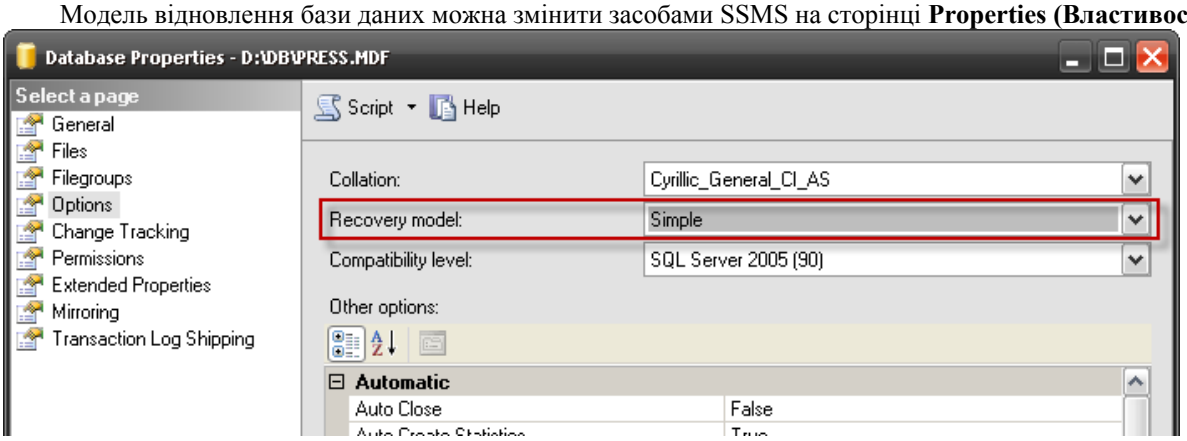
Варто відзначити, що регулярне виконання backup/restore бази даних та журналу транзакцій дозволяє уникнути втрати даних.

Всі типи резервного копіювання, які підтримуються SQL Server 2008, залежать від **моделі відновлення бази даних (recovery model)**. Вони також визначають, як SQL Server буде працювати з журналом транзакцій, буде його реєструвати та очищати (точніше обрізати (truncating) журнал, тобто видаляти зафіксовані транзакції і вивільняти місця для реєстрації нових). SQL Server 2008 підтримує **три моделі відновлення бази даних**:

1. **Модель повного відновлення (Full recovery model)** – в журналі транзакцій будуть реєструватись всі операції і очищення журналу відбуватись не буде. Дана модель дозволяє повністю відновити базу даних до стану на момент аварійного завершення її роботи.
2. **Проста модель відновлення (Simple recovery model)** – реєструє мінімум даних про більшість транзакцій і виконує очищення журналу транзакцій після кожної контрольної точки. Таким чином, формується зручний і компактний журнал транзакцій, який забезпечує доволі високе підвищення продуктивності, але не буде пригодний для відновлення. В зв'язку з цим, дана модель не підтримує резервне копіювання і відновлення журналу транзакцій, а також не дозволяє відновлювати окремі сторінки.
3. **Модель з неповним протоколюванням (Bulk-Logged recovery model)** – являється спрощеною моделлю повного відновлення, в якій про звичайні операції в журнал транзакції записується повна інформація, а от реєстрація інформації про масові операції (SELECT INTO та BULK INSERT) зводиться до мінімуму. Отже, якщо в резервній копії журналу транзакцій містяться які-небудь масові операції, базу даних можна відновити лише до стану, який відповідає кінцю резервної копії журналу, тобто не можна відновити на певний період часу. Ця модель відновлення використовується тільки для великих масових операцій.

На практиці, рекомендується використовувати модель повного відновлення бази даних, оскільки вона володіє найбільшими можливостями відновлення. Якщо ж ви використовуєте масові операції для імпорту даних, краще в такий період змінювати модель відновлення на модель з неповним протоколюванням, оскільки вона дозволяє підвищити продуктивність масових операцій.

Модель відновлення бази даних можна змінити засобами SSMS на сторінці **Properties (Властивості)** бази даних.



А також за допомогою інструкції **ALTER DATABASE** з наступним синтаксисом:

```
-- синтаксис інструкції ALTER DATABASE для зміни моделі відновлення бази даних
ALTER DATABASE ім'я_бази_даних
SET RECORERY { FULL | SIMPLE | BULK_LOGGED }

-- наприклад, встановити повну модель відновлення бази даних
alter database Press
set recovery full;
```

Переглянути інформацію про поточну модель відновлення можна за допомогою системного представлення **sys.databases**:

```
select name, recovery_model_desc
from sys.databases;
```

Результат:

Results | Messages

	name	recovery_model_desc
1	master	SIMPLE
2	tempdb	SIMPLE
3	model	FULL
4	msdb	SIMPLE
5	D:\DB\PRESS.MDF	SIMPLE
6	Test	FULL

6.2. Резервне копіювання

6.2.1. Види резервного копіювання

Резервне копіювання бази даних – це процес зчитування всіх даних з бази даних і збереження їх у вигляді одного чи кількох файлів на диску або пристрої резервного копіювання. Резервні копії можна робити для цілої бази даних або її частини, набору файлів або файлових груп.

Важливо знати, що всі операції резервного копіювання виконуються в контексті безпеки облікового запису користувача, тому такому користувачу необхідно надати відповідні права. До необхідних прав слід віднести права на читання і запис в довільні каталоги або дискові накопичувачі, які будуть використовуватись для створення резервної копії, а також він повинен бути членом фіксованої ролі рівня бази даних **db_backupoperator**.

Існують наступні **види резервного копіювання**:

1. **Повне резервне копіювання бази даних (full database backup)** – дозволяє зберегти поточний стан всіх даних, які зберігаються в базі. Для цього механізм резервного копіювання витягує всі екстенти[1] бази даних, які виділені для об'єктів.
2. **Різничне резервне копіювання бази даних (differential database backup)** – зберігає всі екстенти, які були змінені з моменту останнього повного резервного копіювання (!). Воно дозволяє зменшити кількість резервних копій журналу транзакцій, які необхідно відновити у випадку втрати даних.
3. **Резервне копіювання журналу транзакцій (transaction log)** – дозволяє зберегти активний журнал (active log), тобто дані журналу, які починаються з номера транзакції (LSN), на якому закінчилось попереднє резервне копіювання журналу транзакцій. Після цього зберігаються всі наступні транзакції до тих пір, поки не буде виявлена відкрита транзакція.
4. **Резервне копіювання файлових груп (filegroup backup)** – дозволяє створювати резервне копіювання окремих файлових груп бази даних.

Для виконання різничного резервного копіювання і резервного копіювання журналу транзакцій необхідно створити повну резервну копію. Але відбуваються вони окремо один від одного.

Підсумовуючи відмітимо також наступні **моменти**:

- використання повних резервних копій разом з різничними і резервними копіями журналів транзакцій дозволяють зберегти всю базу даних, включаючи всі зміни, які відбулись з моменту останнього повного резервного копіювання;
- для створення копій фрагментів бази даних краще всього використовувати резервне копіювання файлових груп разом з різничними резервними копіями і копіями журналу транзакцій.

6.2.2. Резервне копіювання бази даних

Механізм резервного копіювання бази даних записує на пристрій резервного копіювання сторінки бази даних, без врахування їх порядку. В зв'язку з цим дані можуть записуватись в кілька потоків, що робить операцію резервного копіювання досить швидкою. Швидкість резервного копіювання залежить лише від швидкості запису даних на пристрій.

Щоб на виході отримати непошкоджені дані (адже під час резервного копіювання до них можуть звертатись інші користувачі і виконувати при цьому вставку, модифікацію або видалення даних), SQL Server **здійснює повне резервне копіювання наступним чином**:

1. Спочатку блокується база даних і всі транзакції.
2. В журнал транзакцій додається маркер.
3. Знімається блокування бази даних.
4. Виконується резервне копіювання всіх сторінок бази даних.
5. Знову здійснюється блокування бази даних і всіх транзакцій.
6. В журнал транзакцій додається маркер.
7. Знімається блокування бази даних.
8. Витягуються всі транзакції між двома маркерами в журналі транзакцій і додаються в резервну копію.

Для резервного копіювання використовується також **карта розміщення екстентів**. Ця карта використовується для визначення екстентів, які повинні ввійти в резервну копію, і являє собою окрему сторінку даних в базі даних, кожен біт якої представляє один екстент. При повному резервному копіюванні всі біти зкидаються на 0. Після цього, якщо екстент змінюється, SQL Server змінює біт, який йому відповідає з 0 на 1. Таким чином, при різничному резервному копіюванні значення бітів карти аналізується і в резервну копію входять лише екстенти, які були змінені (тобто їх біт рівний 1).

Відмітимо, що для бази даних master можна здійснювати лише повне резервне копіювання.

---

[1] **Екстент** – це основна одиниця організації простору в SQL Server. Екстент являє собою колекцію, яка складається з 8 фізичних неперервних сторінок (64 Кб).

Сторінка – це основна одиниця джерела даних в SQL Server. Місце на диску, де розміщується файл бази даних, логічно розділяється на неперервні сторінки по 8 Кб з нумерацією від 0 до n. Всі дискові операції вводу-виводу виконуються на рівні сторінок. Всі сторінки зберігаються в екстентах.

Файли журналу транзакцій не містять сторінок, в них розміщується послідовність записів журналу (!).

Ну і нарешті, резервне копіювання бази даних здійснюється за допомогою оператора **BACKUP DATABASE**:

```
BACKUP DATABASE { база_даних | @змінна_з_іменем_БД }  -- база даних або її частина
                                                      --(об'єкт)
TO пристрій_резервного_копіювання [ ,...n ]
[ MIRROR TO пристрій_резервного_копіювання  ,...n ]] -- дзеркало для основного пристрою
                                                     -- резервного копіювання
-- параметри операцій створення резервної копії
[ WITH [ DIFFERENTIAL ]  -- різнична резервна копія
       [ загальні_опції_WITH [ ,...n ] ]
]

-- пристрій_резервного_копіювання:
{ ім'я_логічного_пристрою_резервного_копіювання
                              | @змінна_імені_логічного_пристрою_резервного_копіювання }
| { DISK | TAPE } = { 'ім'я_фізичного_пристрою_резервного_копіювання'
                        | @змінна_імені_фізичного_пристрою_резервного_копіювання }
```

Згідно вищеописаного синтаксису, спочатку необхідно вказати назву бази даних (або її частини), резервну копію якої необхідно зробити. Після ключового слова **TO** вказується пристрій резервного копіювання, на якому буде зберігатись резервна копія. Ним може бути як ім'я створеного логічного пристрою резервного копіювання, так і повний шлях на диску або стрічковому пристрої.

ПРИМІТКА! В наступній версії SQL Server підтримка стрічкових пристроїв буде вилучена. Тому варто уникати використання параметра TAPE.

Параметр **MIRROR TO** служить для опису пристрою (-їв) резервного копіювання, який буде дзеркалом для основного пристрою резервного копіювання. Максимальна кількість пристроїв для дзеркального резервного копіювання – три. При цьому, вказана кількість пристроїв в параметрі MIRROR TO повинна мати той же тип і ту ж кількість, що і в параметрі TO.

Параметр **WITH** дозволяє задати додаткові необов'язкові параметри створення резервної копії, а також вказати необхідність створення різничної резервної копії бази даних (по замовчуванню створюється повна резервна копія). В якості **загальних опцій параметра WITH** можуть виступати:

I. Параметри резервного набору даних, який створюється поточною операцією резервного копіювання.

Параметр	Опис
COPY_ONLY	Резервна копія лише для копіювання. Якщо параметри DIFFERENTIAL і COPY_ONLY використовуються разом, тоді параметр COPY_ONLY ігнорується і створюється різнична резервна копія.
COMPRESSION | NO_COMPRESSION	Використовується компресія (compression) даної резервної копії.
DESCRIPTION = { 'текст' | @змінна_тексту }	Текстовий опис резервного набору даних, який може містити до 255 символів.
NAME = { 'ім'я' | @змінна_імені }	Ім'я резервного набору даних, яке не повинно перевищувати 128 символів. По замовчуванню ім'я відсутнє.
PASSWORD = { 'пароль' | @змінна_пароля }	Пароль на резервний набір даних. В наступній версії SQL Server ця опція буде видалена.
EXPIREDATE = { 'дата' | @змінна_дати }	Час, який визначає, коли дану резервну копію можна перезаписати. Ці параметри можуть вказуватись разом, але в такому виадку RETAINDAYS має пріоритет над EXPIREDATE. Якщо жоден з цих параметрів не заданий, тоді термін збереження визначається параметром конфігурації mediaretention. Якщо вказані параметри задаються за допомогою змінної, тоді вони повинні мати один з наступних типів: • для EXPIREDATE: - рядковою константою; - символьним рядком, крім типів даних ntext і text; - smalldatetime; - datetime. • для RETAINDAYS – цілим числом.
RETAINDAYS = { к-ть_днів | @змінна_к-ті_днів }	

II. Параметри набору носіїв, на які записуються резервні копії:

Параметр	Опис
NOINIT | INIT	Визначають чи буде операція резервного копіювання перезаписувати (INIT) резервні набори даних, які вже існують на носії резервної копії, чи дописувати (NOINIT) нові набори даних в кінець. По замовчуванню використовується опція NOINIT.
NOSKIP | SKIP	Визначає чи буде операція резервного копіювання перевіряти (NOSKIP) дату і час закінчення терміну резервної копії на носії перед їх перезаписом. Опція NOSKIP встановлена по замовчуванню.
NOFORMAT | FORMAT	Повинен (FORMAT) чи ні (NOFORMAT) заголовок носія записуватись (або перезаписуватись) на томах, які використовуються поточною операцією резервного копіювання. В зв'язку з цим, опцію FORMAT слід використовувати дуже обережно, оскільки форматування будь-якого тому з набору носіїв пошкоджує всю резервну копію.
MEDIADESCRIPTION = { 'текст' | @змінна_тексту }	Текстовий опис набору носіїв, який не повинен перевищувати 255 символів.
MEDIANAME = { 'ім'я_носія' | @змінна_імені }	Ім'я носія для всього набору носіїв резервних копій, яке не повинне бути більше 128 символів. Це ім'я повинно співпадати з вже заданим іменем носія, який існує в томах резервних копій. Якщо ім'я не вказане і вказаний параметр SKIP, тоді перевірки імені носія не буде.
MEDIAPASSWORD = { 'пароль' | @змінна_пароля }	Пароль на набір носіїв. В наступній версії SQL Server даний параметр буде відсутній.
BLOCKSIZE = { розмір | @змінна_розміру }	Розмір фізичного блоку в байтах, який може приймати одне з наступних значень: 512, 1 024, 2 048, 4 096, 8 192, 16 384, 32 768 і 65 536 байт. Значення по замовчуванню - 65536 для стрічкових пристроїв і 512 для інших. На практиці, даний параметр майже не використовується, оскільки оператор BACKUP автоматично обирає розмір необхідного блоку.

III. Параметри передачі даних:

Параметр	Опис
BUFFERCOUNT = { к-ть_буферів | @змінна_к-ті_буферів }	Загальна кількість буферів вводу-виводу, які будуть використовуватись при резервному копіюванні. Відмітимо, що при великій кількості буферів операція резервного копіювання може бути дуже ресурсоємкою. Загальний простір, який використовується буферами визначається по наступній формулі: к-ть_буферів * об'єм_пакета.
MAXTRANSFERSIZE = { об'єм_пакета | @змінна_об'єму_пакета }	Найбільший об'єм пакета даних в байтах для передачі між SQL Server і носієм резервного набору. Підтримуються значення, кратні 65 536 байтам (64 Кб) і до 4 194 304 байт (4 Мб).

IV. Параметри управління помилками:

Параметр	Опис
NO_CHECKSUM | CHECKSUM	Дозволені (CHECKSUM) чи ні (NO_CHECKSUM) контрольні суми. По замовчуванню контрольні суми відсутні, за виключенням зжатих резервних копій.
STOP_ON_ERROR | CONTINUE_AFTER_ERROR	Чи буде операція резервного копіювання зупинена після виявлення помилки в контрольній сумі сторінки (STOP_ON_ERROR) чи продовжити роботу (CONTINUE_AFTER_ERROR). По замовчуванню встановлений STOP_ON_ERROR.

V. Параметри сумісності представлені одним параметром **RESTART**, який нічого не робить і залишений лише для забезпечення зворотної сумісності.

VI. В параметрах спостереження існує також один представник – це параметр **STATS [= відсотковий_інтервал]**. Він дозволяє виводити повідомлення кожен раз по закінченні вказаного відсоткового інтервалу. По замовчуванню повідомлення виводиться після кожних 10 відсотків.

VII. Параметри стрічкових пристроїв:

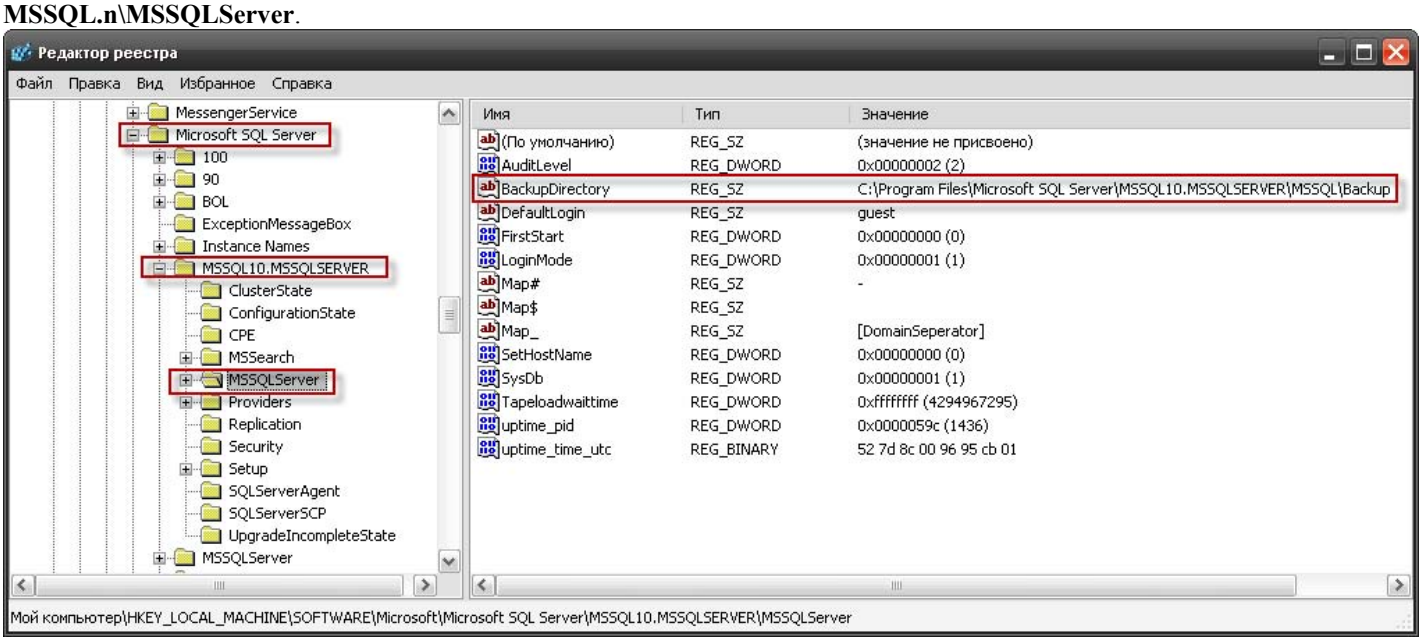
Параметр	Опис
REWIND | NOREWIND	Вивільнити і перемотати стрічку (REWIND; значення по замовчуванню) або ж зберегти її відкритою після закінчення резервного копіювання (NOREWIND), що дозволяє суттєво покращити продуктивність при виконанні кількох операцій резервного копіювання на стрічку. Параметр NOREWIND включає в себе параметр NOUNLOAD, тому разом їх вказувати в операторі BACKUP не дозволяється.
UNLOAD | NOUNLOAD	Дозволяють налаштувати поточний сеанс резервного копіювання: - UNLOAD вказує на те, що стрічка повинна автоматично перемотуватись і вигружатись по завершенні операції backup. - NOUNLOAD вказує на те, що стрічка буде зберігатися в стрічковому накопичувачі після операції backup.

ПРИМІТКА! В символьних параметрах не дозволяється вказувати типи даних ntext і text. Крім того, оператор BACKUP заборонено використовувати в явних і неявних транзакціях.

Наведемо приклад створення повної резервної копії бази даних Press в файл на диску. При цьому всі резервні набори, які є на диску слід перезаписати:

```
backup database Press
to disk = 'D:\Backups\Press.bak'
with init;
```

Файл резервної копії – це звичайний файл і тому достатньо просто прописати його назву з повним шляхом. При створенні резервної копії можна вказувати абсолютний або відносний шлях до файла, а також взагалі його упустити. В двох останніх випадках, буде використовуватись каталог по замовчуванню, яким являється **«C:\Program Files\Microsoft SQL Server\MSSQL.n\MSSQL\Backup»**, де n - номер екземпляра сервера. Каталог по замовчуванню можна змінити в розділі реєстру **BackupDirectory**, в гілці **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Microsoft SQL Server\MSSQL.n\MSSQLServer**.



Крім того, можна звертатись до віддаленого дискового файла. Щоб мати доступ до спільного мережевого ресурсу, необхідно, щоб цей диск був підключений як мережевий. Щоб вказати мережевий ресурс при резервному копіюванні, необхідно використовувати повне ім'я в форматі UNC, яке має наступну форму: **\\Ім'я_системи\Спільна_папка\Шлях\Ім'я_файла**. Наприклад:

```
backup database Press
to disk = '\\BackupServer\Backups\Press.bak';
```

Щодо логічних пристроїв резервного копіювання, то вони використовуються для неявного звернення до існуючих фізичних пристроїв резервного копіювання. Це дозволяє спростити процес вказування файла резервної копії, оскільки відпадає необхідність прописувати довгий повний шлях. Для їх створення використовується зберігаєма процедура **sp_addumpdevice**, яка дозволяє зв'язати фізичний пристрій резервного копіювання з логічним.

```
sp_addumpdevice [ @devtype = ] 'тип_пристрою_резервного_копіювання'      -- disk або tape
        , [ @logicalname = ] 'ім'я_логічного_пристрою_резервного_копіювання'
        , [ @physicalname = ] 'ім'я_фізичного_пристрою_резервного_копіювання'
        -- застарівші параметри, які існують для сумісності
        [, { [ @cntrltype = ] тип_контролера | [ @devstatus = ] 'статус_пристрою' } ]
```

Наприклад, запишемо резервну копію нашої бази даних Press на логічний пристрій резервного копіювання PressBack:

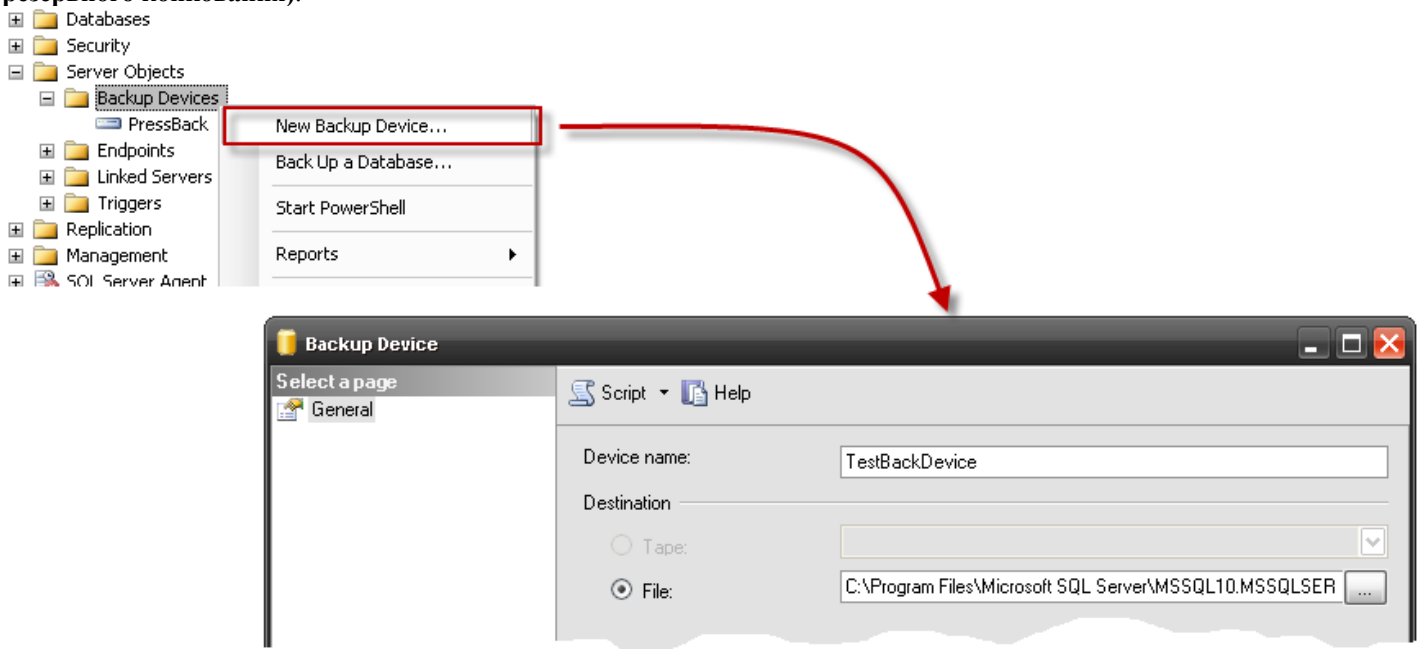
```
-- створюємо логічний пристрій резервного копіювання
exec sp_addumpdevice 'disk', 'PressBack', 'D:\Backups\Press.bak';

-- створюємо резервну копію бази даних на логічний пристрій резервного копіювання
backup database Press
to PressBack;
```

Для видалення існуючого логічного пристрою резервного копіювання використовується системна зберігаєма процедура **sp_dropdevice**:

```
-- синтаксис
sp_dropdevice [ @logicalname = ] 'ім'я_логічного_пристрою_резервного_копіювання'
              [ , [ @delfile = ] 'delfile' ]   -- чи потрібно видаляти
                                               -- фізичний пристрій резервного копіювання

-- наприклад, видалимо один з логічних пристроїв резервного копіювання,
-- разом з фізичним пристроєм, який йому відповідає
exec sp_dropdevice 'PressBack', 'delfile'
```

Для перегляду імен існуючих логічних пристроїв, можна скористатись системним представленням **sys.backup_devices**.

Для створення пристрою резервного копіювання засобами SSMS необхідно обрати в необхідному сервері, в папці **Server Objects** або її підпапці **Backup Devices** пункт контекстного меню **New Backup Device… (Новий пристрій резервного копіювання)**.



Після цього в діалоговому вікні Backup Device необхідно вказати ім'я та шлях до створюваного ресурсу.

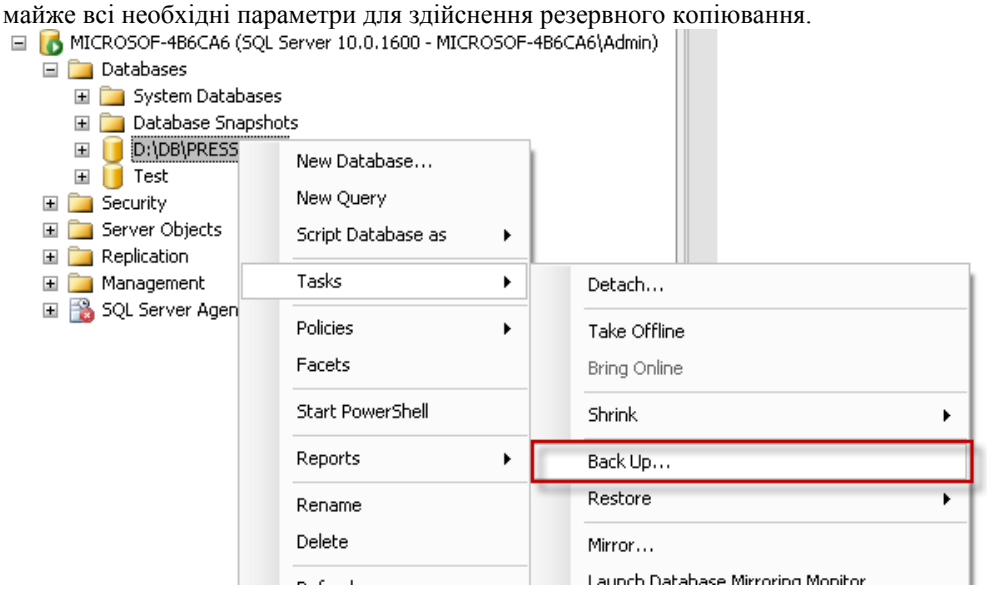
А тепер розглянемо приклад створення **дзеркальних наборів носіїв (media set)**, яке дозволяє створити кілька додаткових копій даних. Існування кількох резервних копій даних ще більше знижує ризик втрати даних у випадку виходу

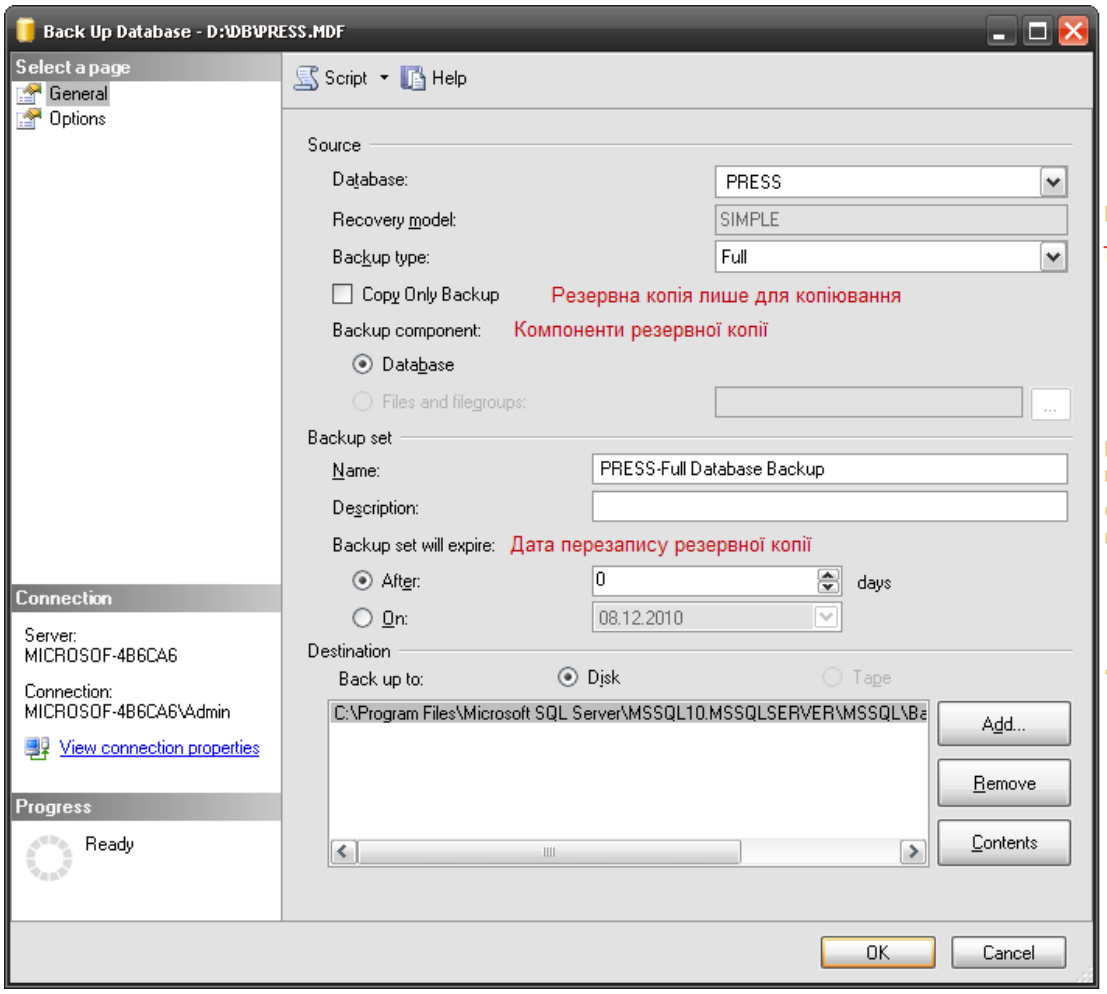
з ладу одного з пристроїв резервного копіювання. При використанні дзеркального резервного копіювання SQL Server зчитує сторінку з файла даних один раз, після чого створює кілька копій при запису сторінки на пристрій резервного копіювання (диск чи стрічку). Таким чином одна сторінка даних записується одночасно в кожне дзеркало.

Отже, збережемо резервну копію бази даних Press на носії, який складається з двох дисків, а також створимо при цьому два дзеркала резервної копії. При цьому резервна копія зберігається на локальний диск, а дзеркала на мережеві. За допомогою опції FORMAT запишемо також нові заголовки носіїв на всі набори носіїв, перезаписуючи попередні заголовки і форматуючи їх вміст. Відмітимо, що без цієї опції створення дзеркальних наборів неможливо. Це обов'язкова умова при дзеркальному резервному копіюванні.

```
BACKUP DATABASE Press
TO DISK = 'D:\Backups\Press1B.bak', DISK = 'D:\Backups\Press2B.bak'
MIRROR TO DISK = '\\BackupServer1\Backups\PressMirror1A.bak',
          DISK = '\\BackupServer\Backups\PressMirror1B.bak'
MIRROR TO DISK = '\\BackupServer2\Backups\PressMirror2A.bak',
          DISK = '\\BackupServer\Backups\PressMirror2B.bak'
WITH
   FORMAT, MEDIANAME = 'PressSet'
```

Резервне копіювання бази даних можна зробити і засобами SSMS. Для цього, в контекстному меню необхідної бази даних необхідно обрати пункт **Tasks->Back Up…** В новому діалоговому вікні **Back Up Database** можна налаштувати майже всі необхідні параметри для здійснення резервного копіювання.





База даних

Модель відновлення

Тип резервного копіювання

Резервна копія лише для копіювання

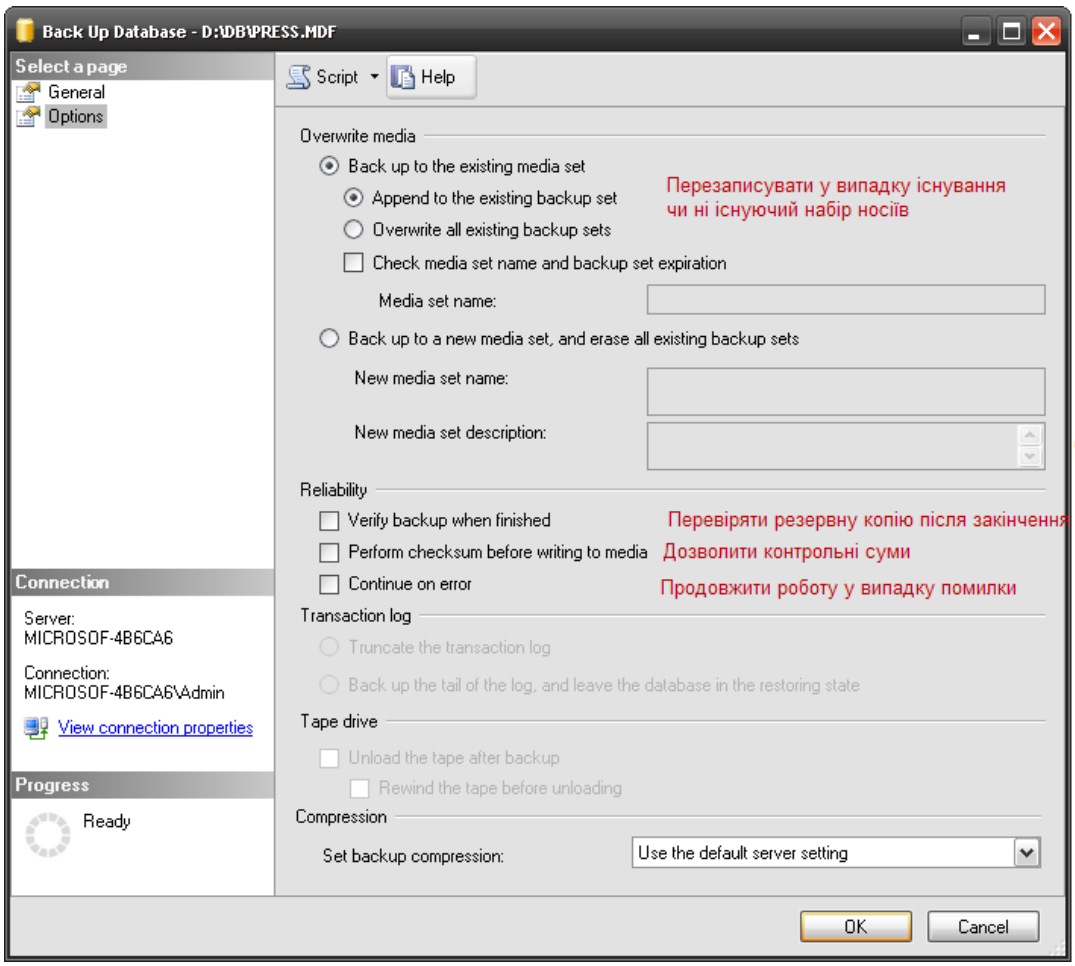
Компоненти резервної копії

Ім'я резервного набору даних

Опис резервного набору даних

Дата перезапису резервної копії

Де зберегти резервну копію



Перезаписувати у випадку існування чи ні існуючий набір носіїв

Ім'я набору носіїв

Ім'я нового набору носіїв

Опис нового набору носіїв

Перевіряти резервну копію після закінчення резервного копіювання

Дозволити контрольні суми

Продовжити роботу у випадку помилки

Для резервного копіювання журналу транзакцій

Для стрічкових пристроїв

Використовувати компресію резервної копії

6.2.3. Резервне копіювання журналу транзакцій

Як вже було сказано, резервне копіювання журналу транзакцій (transaction log) дозволяє зберегти активний журнал (active log). Після цього зберігаються всі наступні транзакції до тих пір, поки не буде виявлена відкрита транзакція.

Резервне копіювання журналу транзакцій можна здійснювати лише для баз даних з **моделлю повного відновлення або моделлю відновлення з неповним протоколюванням**. Крім того, створення резервної копії журналу транзакцій можливе лише після виконання повного резервного копіювання бази даних.

Для резервного копіювання журналу транзакцій використовується інструкція **BACKUP LOG**:

```
BACKUP LOG { база_даних | @змінна_імені_БД }
TO пристрій_резервного_копіювання [ ,...n ] -- див.оператор BACKUP для резервного
                                            -- копіювання бази даних
[ MIRROR TO пристрій_резервного_копіювання [ ,...n ]]
[ WITH { загальні_опції_WITH | LOG_опції }  ]
```

Як видно з синтаксису, інструкція BACKUP LOG майже нічим не відрізняється від інструкції BACKUP DATABASE для створення резервної копії бази даних. **Основна відмінність** полягає в тому, що крім загальних параметрів створення резервної копії, можна вказувати ще ряд параметрів, які відносяться лише до операції резервного копіювання журналу транзакцій. Отже, до LOG параметрів **відносять**:

Параметр	Опис
NORECOVERY | STANDBY = ім'я_резервного_файла	Створюють резервну копію журналу транзакцій і залишають базу даних в стані RESTORING (NORECOVERY) або лише для читання і в стані STANDBY (STANDBY). Для найбільш ефективного створення резервної копії журналів, при якому не відбувається його урізання і база даних автоматично переходить в стан RESTORING, використовуйте одночасно параметр NORECOVERY з NO_TRUNCATE. Використання параметра STANDBY рівносильно параметру BACKUP LOG WITH NORECOVERY, за яким йде RESTORE WITH STANDBY. При використанні режиму очікування необхідно вказати резервний файл, в якому містяться всі зміни, які пройшли відкат і повинні бути відновлені у випадку необхідності.
NO_TRUNCATE	Журнал транзакцій не урізається і резервне копіювання здійснюється не залежно від стану бази даних. В зв'язку з цим, резервна копія може мати неповні метадані. Як правило, даний параметр використовується при пошкодженні бази даних. Параметр NO_TRUNCATE рівносильний одночасному використанню COPY_ONLY і CONTINUE_AFTER_ERROR.

Варто відмітити, що **резервні копії бази даних і журналу транзакцій рекомендується зберігати на різних дисках**. Це необхідно для забезпечення безпеки резервних копій у випадку пошкодження одного з дисків.

Наведемо невеличкий приклад створення повної резервної копії бази даних та журналу транзакцій. В якості пристроїв резервного копіювання використаємо логічні пристрої.

```
-- створюємо логічні пристрої резервного копіювання
-- для збереження резервної копії бази даних
exec sp_addumpdevice 'disk', 'PressData', 'D:\Backups\PressData.bak';
go
-- для збереження резервної копії журналу транзакцій
exec sp_addumpdevice 'disk', 'PressLog', 'D:\Backups\PressLog.trn'; -- розширення довільне
go

-- робимо повну резервну копію бази даних
backup database Press
to PressData;
go
-- робимо резервну копію журналу транзакцій
backup log Press
to PressLog;
go
```

6.2.4. Резервне копіювання файлових груп та створення часткової резервної копії

В SQL Server 2008, крім створення повної резервної копії бази даних, існує можливість **резервного копіювання** окремих **файлових груп (filegroup backup)**. Такий підхід дуже корисний, якщо через розмір бази даних її повне збереження або відновлення стає невиправданим.

Оскільки резервне копіювання файлових груп зберігає лише фрагменти бази даних, для його виконання необхідно, щоб для бази даних була встановлена модель повного відновлення або модель відновлення з неповним протоколюванням. Крім того, якщо необхідно відновити файлові групи з копій, зроблених у різний час, для їх согласованості на певний період часу, необхідна наявність журналу транзакцій.

Для резервного копіювання файлових груп бази даних використовується наступна форма інструкції BACKUP DATABASE, яка практично нічим не відрізняється від створення резервної копії бази даних:

```
BACKUP DATABASE { ім'я_БД | @змінна_імені_БД }
{ FILE = { логічне_ім'я_файлу | @змінна_логічного_імені_файлу }
  | FILEGROUP = { логічне_ім'я_файлової_групи | @ змінна_логічного_імені_файлової_групи}
}  [ ,...n ]
TO пристрій_резервного_копіювання [ ,...n ] -- див.оператор BACKUP для резервного
                                            -- копіювання бази даних
[ MIRROR TO пристрій_резервного_копіювання [ ,...n ]]
[ WITH { DIFFERENTIAL | загальні_опції_WITH [ ,...n ] } ]
```

Наприклад, створимо різничну резервну копію кожного файла, які розміщуються в двох вторинних файлових групах.

```
BACKUP DATABASE Press
       FILEGROUP = 'PressFilegroup1',
       FILEGROUP = 'PressFilegroup2'
TO DISK = 'D:\Backups\PressFilegroups.bak'
WITH DIFFERENTIAL
```

Та існують випадки, коли в базі даних кілька файлових груп доступні лише для читання. В SQL Server 2008 такі файлові групи не будуть входити в резервну копію бази даних, якщо при її створенні встановити параметр **READ_WRITE_FILEGROUPS**. Але можна вказати і виняток. Для цього після параметра READ_WRITE_FILEGROUPS необхідно перелічити необхідні файлові групи для читання, які планується зберегти.

Якщо сама база даних доступна лише для читання, READ_WRITE_FILEGROUPS включає в резервну копію лише первинну файлову групу.

Така резервна копія буде **частковою резервною копією бази даних** і синтаксис її створення приведений нижче.

```
BACKUP DATABASE { ім'я_БД | @змінна_імені_БД }
READ_WRITE_FILEGROUPS
[, FILEGROUP = { логічне_ім'я_файлової_групи
                | @змінна_логічного_імені_файлової_групи [ ,...n ] ] -- файлова група
                                                                     -- тільки для читання
TO пристрій_резервного_копіювання [ ,...n ] -- див.оператор BACKUP для резервного
                                            -- копіювання бази даних
[ MIRROR TO пристрій_резервного_копіювання [ ,...n ]]
[ WITH { DIFFERENTIAL |  загальні_опції_WITH [ ,...n ] } ]
```

Наприклад:

```
BACKUP DATABASE Press
READ_WRITE_FILEGROUPS
TO DISK = 'D:\Backups\PressRW.bak'
```

6.3. Відновлення з резервної копії

Процес відновлення бази даних відбувається згідно обраної вами стратегії відновлення, але в більшості випадків вона розпочинається з повного відновлення бази даних, яка відновлює базу даних на певний період часу. Адже повна резервна копія містить всі дані бази. Після цього використовуються часткові (різничні) послідовні резервні копії, які дозволяють привести її до стану на пізніший період.

Сам процес відновлення з резервної копії довготривалий, адже, якщо база даних відновлюється з нуля, операція відновлення спочатку створює для бази даних всі файли і файлові групи, після чого по порядку відновлюються всі сторінки. Тому, радимо при можливості перед відновленням не видаляти базу даних, а перезаписати її.

Відновлення резервної копії виконується за допомогою оператора **RESTORE**, який дозволяє здійснювати наступні **сценарії відновлення**:

- **Повне відновлення** – відновлює базу даних з повної резервної копії.
- **Часткове відновлення** – відновлює частину базу даних з різничної резервної копії.

- **Відновлення файлів** – відновлює в базі даних вказані файли або файлові групи.
- **Відновлення сторінок** – відновлює вказані сторінки бази даних.
- **Відновлення журналу транзакцій** – відновлює журнал транзакцій бази даних.
- **Відновлення по знімку** - дозволяє повернути базу даних до моменту створення знімку (snapshot) бази даних.

При цьому, якщо відновлення резервних копій журналу транзакцій не потрібне, слід використовувати просту модель відновлення.

Приведемо синтаксис оператора **RESTORE** для перелічених вище випадків використання.

```
-- для повного відновлення бази даних
RESTORE DATABASE { ім'я_БД | @змінна_імені_БД }
[ FROM пристрій_резервного_копіювання [ ,...n ] ] -- див.оператор BACKUP для резервного
                                                  -- копіювання БД
[ WITH
 {
      [ RECOVERY | NORECOVERY
        | STANDBY = {standby_file_name | @standby_file_name_var } ] -- залишає БД
                                                -- в режимі відновлення, тобто дозволяється
                                                -- здійснювати лише вибірку даних
      |, загальні_опції_WITH [ ,...n ]
      |, KEEP_REPLICATION     -- опція_реплікації_WITH
      |, KEEP_CDC             -- опції_WITH_відслідковування_змін_даних
      |, опції_WITH_service_broker
      |, опції_WITH_на_заданий_момент_часу
 } [ ,...n ]
]

-- для часткового відновлення бази даних
RESTORE DATABASE { ім'я_БД | @змінна_імені_БД }
{
   FILE = { логічне_ім'я_файлу | @змінна_логічного_імені_файлу }
   | FILEGROUP = { логічне_ім'я_файлової_групи | @змінна_логічного_імені_файлової_групи}
   | READ_WRITE_FILEGROUPS
} [ ,...n ]
[ FROM пристрій_резервного_копіювання [ ,...n ] ]
WITH
      PARTIAL,    -- відновити первинну і вказані вторинні файлові групи
      NORECOVERY
      [, загальні_опції_WITH | опції_WITH_на_заданий_момент_часу ] [ ,...n ]

-- для відновлення файлів та файлових груп
RESTORE DATABASE { ім'я_БД | @змінна_імені_БД }
{
   FILE = { логічне_ім'я_файлу | @змінна_логічного_імені_файлу }
   | FILEGROUP = { логічне_ім'я_файлової_групи | @змінна_логічного_імені_файлової_групи}
   | READ_WRITE_FILEGROUPS
} [ ,...n ]
[ FROM пристрій_резервного_копіювання [ ,...n ] ]
WITH { [ RECOVERY | NORECOVERY ]
       [, загальні_опції_WITH [ ,...n ] ]
     } [ ,...n ]

-- для відновлення сторінок бази даних
RESTORE DATABASE { ім'я_БД | @змінна_імені_БД }
PAGE = 'Id_файла:Id_сторінки [ ,...n ]' -- список сторінок для відновлення
                                        -- (максимум 1000 сторінок)
[, { FILE = { логічне_ім'я_файлу | @змінна_логічного_імені_файлу }
   | FILEGROUP = { логічне_ім'я_файлової_групи | @змінна_логічного_імені_файлової_групи}
   | READ_WRITE_FILEGROUPS
   }
] [ ,...n ]
[ FROM пристрій_резервного_копіювання [ ,...n ] ]
WITH NORECOVERY
```

```
     [, загальні_опції_WITH [ ,...n ] ]

-- для відновлення журналів транзакцій
RESTORE LOG { ім'я_БД | @змінна_імені_БД }
[ { FILE = { логічне_ім'я_файлу | @змінна_логічного_імені_файлу }
   | FILEGROUP = { логічне_ім'я_файлової_групи | @змінна_логічного_імені_файлової_групи}
   | READ_WRITE_FILEGROUPS
   | PAGE = 'Id_файла:Id_сторінки [ ,...n ]'
  } [ ,...n ]
]
[ FROM пристрій_резервного_копіювання [ ,...n ] ]
[ WITH
  {
    [ RECOVERY | NORECOVERY | STANDBY = {ім'я_standby_файла
                                        | @змінна_імені_standby_файла } ]
    |, загальні_опції_WITH [ ,...n ]
    |, KEEP_REPLICATION  -- опція_реплікації_WITH
    |, опції_WITH_на_заданий_момент_часу
  } [ ,...n ]
]

-- повернути базу даних на момент створення знімку (snapshot)
RESTORE DATABASE { ім'я_БД | @змінна_імені_БД }
FROM DATABASE_SNAPSHOT = ім'я_знімка_БД
```

Параметр **WITH** дозволяє задати додаткові необов'язкові параметри відновлення з резервної копії. Ряд загальних параметрів WITH збігається з параметрами WITH для операції BACKUP (див. п.6.2.2. «Резервне копіювання бази даних»). До них відносять:

- Параметри набору носіїв (Media Set Options);
- Параметри передачі даних (Data Transfer Options);
- Параметри управління помилками (Error Management Options);
- Параметри спостереження (Monitoring Options);
- Параметри стрічкових пристроїв (Tape Options).

В SQL Server 2008 параметр **PARTIAL** запускає початковий етап поетапного відновлення, яке дозволяє відкласти відновлення файлових груп, які залишились, тобто не були вказані при відновленні.

Ще кілька загальних параметрів WITH приведено нижче:

Параметр	Опис
Параметри операції відновлення (Restore Operation Options)	
MOVE 'логічне_ім'я_в_backup' TO 'ім'я_файла_з_шляхом' [,...n]	Вказує на те, що файл даних або журнал транзакцій, який вказаний після MOVE, необхідно відновити з копії і перемістити по адресі, заданій в параметрі TO.
REPLACE	Якщо при відновленні з резервної копії база даних (або журнал транзакцій) з аналогічним іменем вже існує, тоді необхідно її перезаписати. Якщо даний параметр не вказаний і при відновленні з резервної копії на сервері вже існує база даних (або журнал транзакцій) з аналогічним іменем, тоді операція відновлення буде відмінена.
RESTART	Перезапустити перервану операцію відновлення з точки переривання.
RESTRICTED_USER	Обмежити доступ до створеної бази даних для членів ролей db_owner, dbcreator і sysadmin. Параметр RESTRICTED_USER являється альтернативою параметра DBO_ONLY ранніх версій SQL Server. Крім того, даний параметр слід використовувати з параметром RECOVERY.
Параметри резервного набору даних (Backup Set Options)	
FILE = { номер_резервного_набору |@змінна_номеру_резервного_набору}	Вказує на резервний набір даних для відновлення. Наприклад, значення 1 вказує на те, що буде використовуватись перший резервний набір даних на носії даних резервних копій, значення 2 вказує на другий набір даних і т.д. Перелік резервних наборів даних можна отримати за допомогою інструкції RESTORE HEADERONLY. По замовчуванню приймається значення -1.

PASSWORD = { 'пароль' | @змінна_пароля }	Пароль на резервний набір даних. В наступній версії SQL Server ця опція буде видалена (!).

Якщо під час створення резервної копії проводилась реплікація бази даних, тоді при відновленні варто вказати **параметр реплікації KEEP_REPLICATION**. Він запобігає видаленню параметрів реплікації, якщо резервна копія бази даних або журналу транзакцій відновлюється на сервері «гарячого» резервування і база даних відтворюється. При цьому, забороняється використовувати даний параметр разом з параметром NORECOVERY.

Якщо при створенні резервної копії в базі даних била ввімкнена система відслідковування змін даних, тоді при відновленні слід вказати ключове слово **KEEP_CDC**. В такому разі SQL Server запобігає видаленню налаштувань системи відслідковування змін даних. Даний параметр також забороняється використовувати разом з параметром NORECOVERY.

Якщо при створенні резервної копії для бази даних був активований **компонент Service Broker**, при відновленні ви можете скористатись **набором опцій WITH** для даного компонента.

Параметр	Опис
ENABLE_BROKER	Після закінчення процесу відновлення ввімкнути доставку повідомлень компонента Service Broker. По замовчуванню при відновленні доставка повідомлень вимкнена. При цьому існуючий ідентифікатор компонента Service Broker зберігається в базі даних.
ERROR_BROKER_CONVERSATIONS	Завершує всі діалоги, які знаходяться в стані помилки і були або приєднані до бази даних, або відновлені. Це дозволяє додаткам виконувати регулярне очищення існуючих сеансів зв'язку. Доставка повідомлень компонента Service Broker до завершення даної операції вимикається, а потім знову вмикається. При цьому існуючий ідентифікатор компонента Service Broker зберігається в базі даних.
NEW_BROKER	Назначає базі даних новий ідентифікатор компонента Service Broker. При цьому всі існуючі сеанси зв'язку в базі даних будуть негайно видалені, без видачі діалогових вікон при завершенні. В зв'язку з цим, всі маршрути, які ссилаються на попередній ідентифікатор компонента Service Broker, потрібно перестворити.

Опції WITH на заданий момент часу дозволяють відновити базу даних на певний момент часу або до певної транзакції, вказавши при цьому необхідну точку відновлення в інструкції STOPAT, STOPATMARK або STOPBEFOREMARK.

Для відновлення на певний момент часу спочатку потрібно відновити повну резервну копію бази даних, кінцева точка якої розміщується раніше цільової точки відновлення. Щоб спростити цей процес, можна ще при створенні резервної копії, в операторі RESTORE DATABASE вказати інструкцію WITH STOPAT, STOPATMARK або STOPBEFOREMARK.

Параметри WITH на заданий момент часу для інструкцій RESTORE_DATABASE і RESTORE_LOG однакові, за виключенням значення параметра STOPATMARK.

Параметр	Опис
STOPAT = { 'дата_і_час' | @змінна_дати_і_часу }	Відновити базу даних або журнал транзакцій до вказаного періоду часу. При цьому до бази даних застосовуються лише ті записи журналу транзакцій, які були зроблені до вказаної дати і часу. Тип даних змінної повинен бути varchar, char, smalldatetime або datetime. (!) Якщо вказаний час назначений після створення останньої резервної копії журналів, база данных залишається у невідновленому стані. Це рівносильно інструкції RESTORE LOG з параметром NORECOVERY.
Для RESTORE DATABASE: STOPATMARK = { 'lsn:lsn_номер' } [AFTER 'дата_і_час'] **Для RESTORE LOG:** STOPATMARK = { 'точка_транзакції' | 'lsn:lsn_номер' } [AFTER 'дата_і_час']	Відновлення до вказаної точки відновлення. Вказана транзакція включається в відновлення, але фіксується лише у випадку, якщо вона була з самого початку зафіксована в ході формування транзакції. Параметр **lsn_номер** визначає реєстраційний номер транзакції в журналі. Якщо в інструкції RESTORE LOG не заданий параметр AFTER, тоді відновлення відбувається до першої точки з вказаним іменем. Інакше відновлення відбувається до першої точки з вказаним іменем перед або після вказаної дати і часу. (!) Якщо вказана точка збереження транзакції, номер LSN або час назначені після створення останньої резервної копії журналів, база даних залишається у невідновленому стані. Це рівносильно інструкції RESTORE LOG з параметром NORECOVERY.

Для RESTORE DATABASE: STOPBEFOREMARK = { 'lsn:lsn_номер' } [AFTER 'дата_і_час'] **Для RESTORE LOG:** STOPBEFOREMARK = { 'точка_транзакції' | 'lsn:lsn_номер' } [AFTER 'дата_і_час']	Відновлення до вказаної точки відновлення. Задана транзакція не включається у відновлення, оскільки після використання параметра WITH RECOVERY відбувається її відкат. Параметр **lsn_номер** визначає реєстраційний номер транзакції в журналі. Якщо в інструкції RESTORE LOG не заданий параметр AFTER, тоді відновлення відбувається до першої точки з вказаним іменем. Інакше відновлення відбувається до першої точки з вказаним іменем перед або після вказаної дати і часу. (!) Відновлення на заданий момент часу не буде виконуватись, якщо послідовність часткового відновлення включає файлові групи FILESTREAM. Але існує можливість примусово продовжити послідовність відновлення. Для цього необхідно вказати параметр CONTINUE_AFTER_ERROR разом з параметром STOPAT, STOPATMARK або STOPBEFOREMARK. Це дозволить виконати послідовність часткового відновлення, але файлова група FILESTREAM при цьому не відновиться.

В SQL Server 2008 існує можливість повернути базу даних до зробленого **знімка бази даних (snapshot)**. Для цього необхідно задати ім'я необхідного знімка в параметрі **DATABASE_SNAPSHOT** при відновленні з резервної копії. Але, слід пам'ятати:

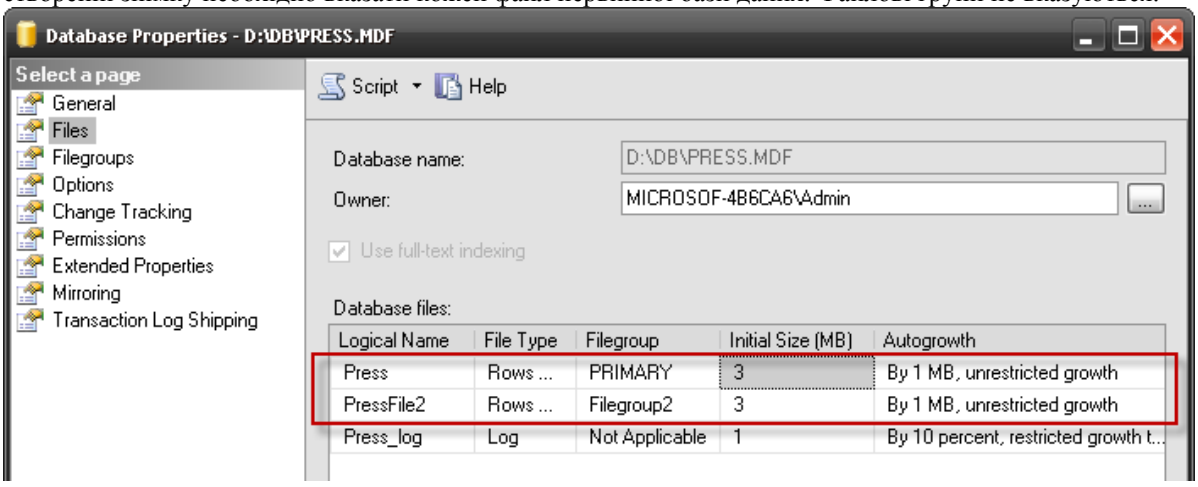
- аргумент DATABASE_SNAPSHOT доступний лише для повного відновлення бази даних;
- при такому відновленні будуть видалені всі повнотекстові каталоги;
- максимальний розмір знімка бази даних рівний розміру самої бази даних;
- розширення файла знімка бази даних може використовуватись довільне.

Приведемо кілька прикладів відновлення:

1. Відновлення бази даних з повної резервної копії:

```
restore database Press
from disk = 'D:\Backups\Press.bak'

-- якщо після відновлення її потрібно перемістити, тоді використаємо наступний запит:
restore database Press
from disk = 'D:\Backups\Press.bak'
with
move 'Press_Data'
to 'C:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\Data\Press.mdf',
move 'Press_Log'
to 'C:\Program Files\Microsoft SQL Server\MSSQL10.MSSQLSERVER\MSSQL\Data\Press.ldf'
```

2. Створимо знімок бази даних Press з іменем Press_sn, після чого спробуємо здійснити повне відновлення бази даних з зробленого знімка.
   Нагадаємо, що наша база даних тепер складається з двох файлів: Press та PressFile2. У відповідності до синтаксису, при створенні знімку необхідно вказати кожен файл первинної бази даних. Файлові групи не вказуються.



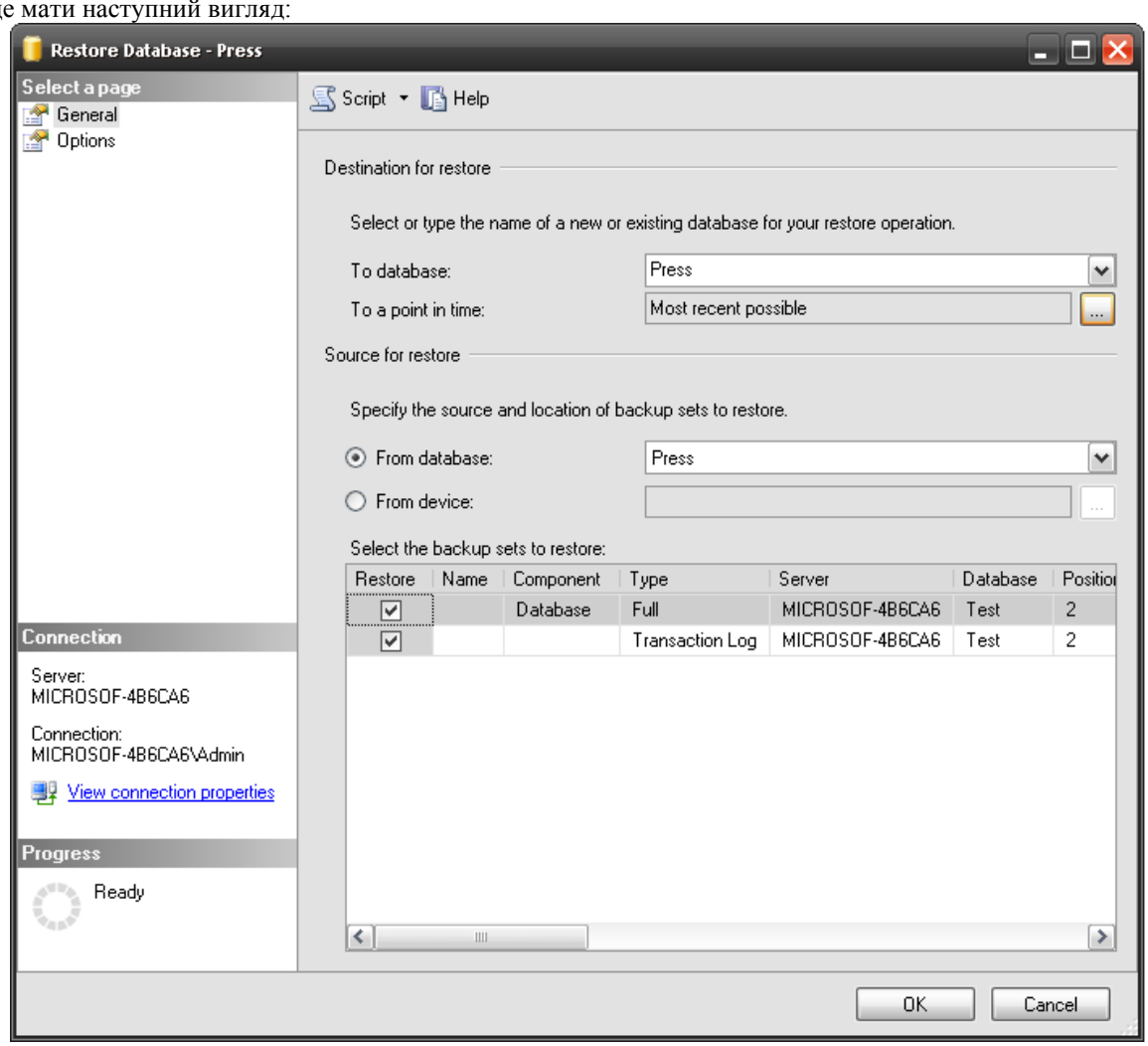
```
-- синтаксис для створення моментального знімка бази даних
CREATE DATABASE ім'я_знімка_БД
ON
      -- список файлів в основній базі даних
      ( NAME = логічне_ім'я_файла,
        FILENAME = 'місцерозташування_файла'
      ) [ ,...n ]
AS SNAPSHOT OF ім'я_основної_БД

-- створюємо знімок бази даних
create database Press_sn
on
( name = Press, filename = 'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\Data\Pressdat1_sn.ss'),
( name = PressFile2, filename = 'C:\Program Files\Microsoft SQL
Server\MSSQL10.MSSQLSERVER\MSSQL\Data\Pressdat2_sn.ss')
as snapshot of Press;
go

-- відновлюємо базу даних з знімку
restore database Press
from database_snapshot = 'Press_sn';
```

4. Відновимо журнал транзакцій до певної мітки, тобто точки збереження в транзакції PointRestore.

```
-- відмічаємо транзакцію, до якої буде відновлений журнал транзакцій
begin transaction PointRestore
with mark 'Deleting old books'; -- вказує на те, що транзакція відмічена в журналі
go
delete
from sale.Sales
where ID_BOOK in
            (select ID_BOOK
            from book.Books
            where DATEPART(YEAR, DateOfPublish) < 2000)
go
delete
from book.Books
where DATEPART(YEAR, DateOfPublish) < 2000
go
commit transaction PointRestore;
go

-- здійснюємо резервне копіювання бази даних та журналу транзакцій
use master;
go
backup database Press
to disk = 'D:\Backups\Press.bak'
with init;
go
backup log Press
to disk = 'D:\Backups\PressLog.trn';
go

-- відновлюємо базу даних та журнал тразакцій
restore database Press
from disk = 'D:\Backups\Press.bak'
with norecovery;
go
restore log Press
from disk = 'D:\Backups\Press.bak'
with recovery,
     stopatmark = 'PointRestore';
```

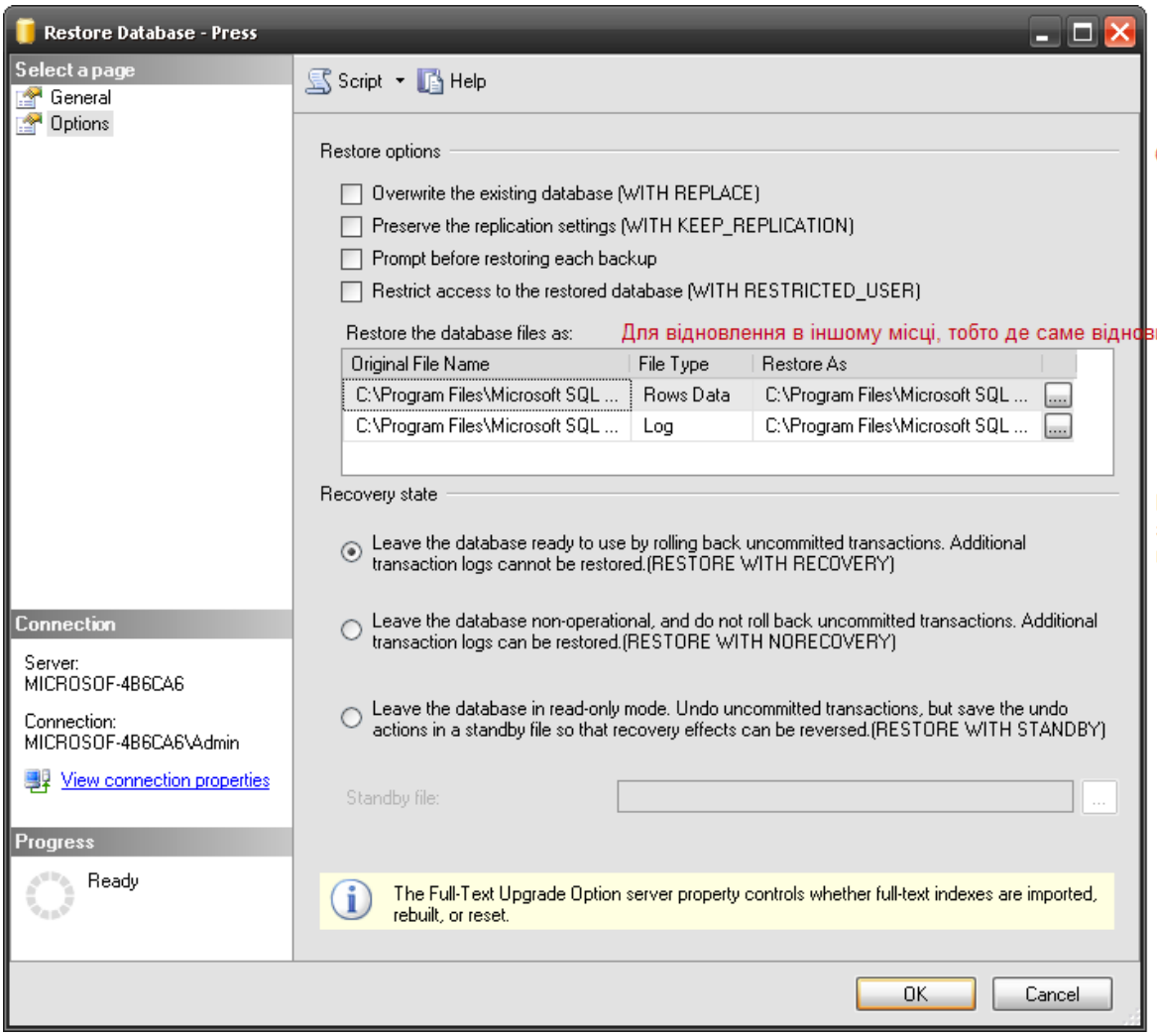
```
go
```

Засобами SSMS операція відновлення дуже схожа з операцією резервного копіювання. Тобто потрібно перейти до бази даних, яку необхідно відновити, і в її контекстному меню обрати пункт **Tasks->Restore**, після чого вказати, що саме потребує відновлення: база даних, журнал транзакцій, файли чи файлові групи. Наприклад, для бази даних, вікно Restore буде мати наступний вигляд:



Параметри збереження відновленої копії

Параметри резервної копії, яку необхідно відновити



6.4. Перевірка резервної копії

Вміти створити резервну копію, нажаль, не достатньо. Адже, резервна копія може пошкодитись або ще в процесі створення, або з часом. В такому разі вона буде не придатна для роботи. Отже, ви повинні також знати як перевірити роботоздатність резервної копії. З однієї сторони, тут нічого складного немає: достатньо просто її відновити і перевірити всі дані. Але цей процес може бути довготривалим, а отже непрактичним. SQL Server 2008 має простіший спосіб вирішення даної проблеми – це інструкція **RESTORE VERIFYONLY**, яка має наступний синтаксис:

```
RESTORE VERIFYONLY
FROM пристрій_резервного_копіювання [ ,...n ] -- див.оператор BACKUP для резервного
                                              -- копіювання БД
[ WITH
 {
      LOADHISTORY -- завантажувати дані в журнальні таблиці резервного копіювання і
                  -- відновлення БД msdb
      -- Параметри операції відновлення
      | MOVE 'логічне_імя_backup_файла' TO 'місцерозташування' [ ,...n ]

      -- Параметри резервного набору даних
      | FILE = { номер_резервного_набору |@змінна_номеру_резервного_набору }
      | PASSWORD = { пароль | @змінна_пароля }

      -- Параметри набору носіїв
      | MEDIANAME = { 'ім'я_носія' | @змінна_імені }
      | MEDIAPASSWORD = { 'пароль' | @змінна_пароля }

      -- Параметри управління помилками
      | { CHECKSUM | NO_CHECKSUM }
      | { STOP_ON_ERROR | CONTINUE_AFTER_ERROR }
```

```
      -- Параметри спостереження
      | STATS [ = відсотковий_інтервал ]

      -- Параметри стрічкових пристроїв
      | { REWIND | NOREWIND }
      | { UNLOAD | NOUNLOAD }
} [ ,...n ]
]
```

Щоб впевнитись в тому, що носій не пошкоджений, дана інструкція перевіряє спочатку його заголовок, потім контрольну суму резервної копії. Після цього зчитується внутрішня послідовність сторінок і перераховується контрольна сума резервної копії для порівняння. Також проводяться різноманітні тести для перевірки цілісності резервної копії, але структура її даних при цьому не зачіпається.

7. Домашнє завдання

Користувачі та права:

1. За допомогою операторів T-SQL створіть ім'я входу з іменем AndreWadi та паролем «mega».
2. За допомогою операторів T-SQL створіть ім'я входу з іменем VladlenKopytych, паролем «superpassword» та базою даних вашого видавництва по замовчуванню. Додати також можливість зміни пароля при першому підключення до SQL Server. Після цього ввімкніть явно дане ім'я входу.
3. Змініть пароль імені входу AndreWadi на наперед хешоване значення, базу даних по замовчуванню на базу даних master, а мову по замовчуванню встановіть англійську.
4. Створіть для імені входу VladlenKopytych двох користувачів бази даних: Vladlen та VladlenKopytych, а для імені входу AndreWadi користувача з аналогічним іменем. Для користувача VladlenKopytych встановити схему по замовчуванню sale. **Примітка!** Користувачі створюються для бази даних вашого видавництва.
5. Зробіть користувача AndreWadi власником вашої бази даних видавництва, але встановіть заборону на управління даними таблиць і представлень.
6. Користувачу Vladlen змініть ім'я на VladlenU і встановіть схему по замовчуванню book. Додайте його до фіксованої серверної ролі dbcreator, а також надайте йому наступні права на базу даних видавництва:
   - всі права на таблицю Books з правом передачі отриманих прав іншим користувачам;
   - права на створення та модифікацію представлень;
   - право REFERENCES на поле ID_BOOK в представленні, яке відображає всю інформацію про роботу магазинів.

7. Створити роль бази даних SAdmin, власником якої є користувач VladlenKopytych. Крім того, вона належить фіксованій ролі бази даних db_securityadmin та має права на перегляд метаданих таблиць, представлень, процедур або функцій.
8. Визначіть права доступу до бази даних для продавців, бухгалтера, менеджера та директора, щоб вони могли виконувати свою роботу. При роздачі прав не забувайте також про наступні моменти:
   - Продавцям необхідно надати набір прав, які дозволять їм не лише продавати книги, але і додавати та змінювати дані про книги, тематики та авторів.
   - Бухгалтеру необхідно крім отримання повної інформації про книги та їх продаж за останній тиждень, надати всі необхідні права на зміну.
   - Менеджер повинен бачити всю поточну картину у видавництві, яка стосується книг, тобто всю інформацію про них, а також мати можливість повністю нею управляти. Справи продажу книг його не стосуються.
   - Директор повинен бачити всю інформацію про випуск книг за поточний тиждень, загальну суму отриманих грошей від продажу книг за поточний місяць та доступ на видалення магазинів.

   У випадку необхідності, створіть потребуємий набір представлень або зберігаємих процедур.

9. Створити роль додатку Sales з довільним паролем та схемою по замовчуванню sale. Після цього активізувати роль, створивши при цьому cookie файл. Крім того, надати ролі додатку Sales наступний набір прав для роботи з вашою базою даних:
   - право на запуск зберігаємої процедури, наприклад, sp_ListAuthors;
   - права UPDATE та DELETE на таблиці book.Books та sale.Shops;
   - право UPDATE на поля Price та Quantity таблиці sale.Sales.

10. Виведіть на екран наступні звіти:
    - звіт про всі права, які має користувач VladlenU та створені вами ролі рівня бази даних (для продавців, бухгалтерів, менеджерів та директора);
    - список прав лише на інструкції в поточній базі даних з використанням NULL в якості заповнювача пропущених параметрів;
    - список всіх прав на поточний сервер.

Резервне копіювання та відновлення з резервної копії:

1. Засобами T-SQL встановіть явно для вашої бази даних модель відновлення з неповним протоколюванням. Після цього створіть та протестуйте стратегію резервного копіювання вашої бази даних, яка включає в себе повне резервне копіюваня, різничне резервне копіювання та резервне копіювання журналу транзакцій. При цьому скористайтесь набором логічних пристроїв резервного копіювання. Після цього перевірте створені резервні копії, відновивши їх.
2. Створіть та протестуйте стратегію резервного копіювання вашої бази даних, яка включає в себе резервне копіювання файлових груп, різничне резервне копіювання та резервне копіювання журналу транзакцій. Після цього перевірте створені резервні копії, відновивши їх.
3. Створіть базу даних, яка містить файлові групи лише для читання і для читання та запису. Виконайте резервне копіювання лише файлових груп для читання і запису.
4. Створіть дзеркальний набір носіїв, який складається з одного диска та чотирьох дзеркал. Після цього напишіть сценарій, який буде здійснювати резервне копіювання бази даних та зберігатись на створеному наборі носіїв.
5. Написати сценарій, який дозволить відновити базу даних до моменту на 9:00 10 листопада 2010 р. Перевірте чи ваша базаних не пошкоджена і доступна для роботи.
6. Створіть для бази даних знімок (snapshot), після чого внесіть до неї ряд змін. Відновіть базу даних, використовуючи створений раніше знімок і перевірте чи існують в ній внесені зміни.
7. Змоделюйте ситуацію, в якій необхідно відновити пошкоджену сторінку. Відновіть в базу даних необхідну сторінку.
8. Припустимо, що у вашій базі даних необхідно провести певне обновлення даних:
    - збільшити ціну продажу всіх книг, які продаються магазинами Англії, на 15%;
    - списати ряд книг, шляхом видалення їх з бази даних, а саме книги, тематики «Детективи», оскільки ваше видавництво більше не буде займатись друком та розповсюдженням таких книг.

    Розробіть та протестуйте стратегію резервного копіювання вашої бази даних, яка забезпечить у випадку необхідності відновлення з резервної копії бази даних до внесення вищеописаних змін. Після цього перевірте створену резервну копію, відновивши її.